Journal of Language Modelling

VOLUME 4 ISSUE 1
JUNE 2016

Institute of Computer Science
Polish Academy of Sciences
Warsaw

Journal of Language Modelling

VOLUME 4 ISSUE 1
JUNE 2016

Editorials

Articles

JOURNAL OF
LANGUAGE MODELLING

ISSN 2299-8470 (electronic version)
ISSN 2299-856X (printed version)
http://jlm.ipipan.waw.pl/

Published by IPI PAN
Institute of Computer Science, Polish Academy of Sciences
ul. Jana Kazimierza 5, 01-248 Warszawa, Poland

Circulation: 100 + print on demand

Layout designed by Adam Twardoch.
Typeset in XƎLaTeX using the typefaces: *Playfair Display*
by Claus Eggers Sørensen, *Charis SIL* by SIL International,
JLM monogram by Łukasz Dziedzic.

EDITORIAL BOARD

Parsing and finite-state technologies, introduction to the special issue

Mark-Jan Nederhof[1] *and Khalil Sima'an*[2]
[1] School of Computer Science, University of St Andrews, UK
[2] Institute for Logic, Language and Computation,
University of Amsterdam, The Netherlands

This issue is dedicated to extended versions of papers published in the proceedings of two conferences. The 11th International Conference on Finite-State Methods and Natural Language Processing was held in July 2013 in St Andrews, Scotland (UK). The 13th International Conference on Parsing Technologies was held in November 2013 in Nara, Japan.

The paper "ZeuScansion: A tool for scansion of English poetry" by Manex Agirrezabal, Mans Hulden, Bertol Arrieta and Aitzol Astigarraga is about scansion, which is the act of marking stressed and unstressed elements in a line of verse and dividing the line into metrical feet. Novel finite-state technology is presented to perform metrical scansion on English poetry.

The paper "On regular languages over power sets" by Tim Fernando is about alphabets that are power sets of finite sets, motivated by, among other things, temporal semantics. Studied are extensions of regular expressions and sentences of monadic second-order logic, offering succinct descriptions of regular languages.

The paper "Data-oriented parsing with discontinuous constituents and function tags" by Andreas van Cranenburgh, Remko Scha and Rens Bod presents an extension of the data-oriented parsing approach for dealing with discontinuous constituents. Two versions are presented, one based on Discontinuous Tree-Substitution Grammars and another based on encoding the discontinuities in the labels of the treebank trees before extracting a Context-Free Grammar.

The paper "On different approaches to syntactic analysis into bi-lexical dependencies: An empirical comparison of direct, PCFG-based, and HPSG-based parsers" by Angelina Ivanova, Stephan Oepen, Rebecca Dridan, Dan Flickinger, Lilja Øvrelid and Emanuele Lapponi presents a comparison of three different approaches to parsing into syntactic, bi-lexical dependencies for English. The approaches consist of a 'direct' data-driven dependency parser, a statistical phrase structure parser, and a hybrid, 'deep' grammar-driven parser. The paper provides extensive analysis of the parsing results of the three approaches being compared.

We would like to thank the authors for contributing to this special issue and the referees for their careful reading of the manuscripts and their helpful reports.

ZeuScansion:
A tool for scansion of English poetry

Manex Agirrezabal[1], *Aitzol Astigarraga*[1],
Bertol Arrieta[1], *and Mans Hulden*[2]
[1] University of the Basque Country (UPV/EHU),
Department of Computer Science, 20018 Donostia, Spain
[2] University of Colorado Boulder, Department of Linguistics,
Boulder, Colorado (USA)

ABSTRACT

We present a finite-state technology (FST) based system capable of performing metrical scansion of verse written in English. Scansion is the traditional task of analyzing the lines of a poem, marking the stressed and non-stressed elements and dividing the line into metrical feet. The system's workflow is composed of several subtasks designed around finite-state machines that analyze verse by performing tokenization, part-of-speech tagging, stress placement, and stress-pattern prediction for unknown words. The scanner also classifies poems according to the predominant type of metrical foot found. We present a brief evaluation of the system using a gold standard corpus of human-scanned verse, on which a per-syllable accuracy of 86.78% is achieved. The program uses open-source components and is released under the GNU GPL license.[1]

Keywords: scansion, English, poetry, out-of-vocabulary words

1 INTRODUCTION

Scansion is a well-established form of poetry analysis which involves marking the prosodic meter of lines of verse and possibly also dividing the lines into feet. The specific technique and scansion notation may

[1] ZeuScansion code:
`https://github.com/manexagirrezabal/zeuscansion`
Stress guesser code: `https://github.com/manexagirrezabal/athenarhythm`

differ from language to language because of phonological and prosodic differences, and also because of different traditions regarding meter and form. Scansion is traditionally done manually by students and scholars of poetry. In the following, we present ZeuScansion, an FST-based software tool for performing this task for English poetry, and provide a brief evaluation of its performance on a gold standard corpus of poetry in various meters.

1.1 *Scansion*

Conventionally, scanning a line of poetry should yield a representation which marks every syllable with its level of stress and divides groups of syllables into units of feet. Typically two or more levels of stress are used.

Consider, for example, the following line from John Keats' poem *To autumn* (Robertson 2007, p. 137).

To swell the gourd, and plump the hazel shells

Here, a natural analysis is as follows:

```
 -    '       -    '     -      '       -   '  -     '
To swell |the gourd |and plump |the haz|el shells
```

We use the symbol `'` to denote stressed (ictic) syllables, and `-` to denote unstressed (non-ictic) ones. That is, we have analyzed the line in question as following the stress pattern

DE-DUM DE-DUM DE-DUM DE-DUM DE-DUM

and also as consisting of five feet of two syllables each with an unstressed–stressed pattern. Indeed, this is the most common meter in English poetry: *iambic pentameter.*

The above example is rather clear-cut. How a particular line of verse *should* be scanned, however, is often a matter of contention. Consider a line from the poem *Le Monocle de Mon Oncle* by Wallace Stevens (1923):

I wish that I might be a thinking stone

Here, matters are much more murky. This line can, for example, be analyzed as five iambic feet,[2] or as one iamb, followed by a pyrrhic

[2] Iambic foot: An unstressed syllable followed by a stressed syllable [`-'`].

foot,[3] followed by two stressed syllables, followed by two more iambs. The following represents several analyses of the line in question.

```
Examp.: I wish that I might be a thinking stone
1st:    -  '    -   '   -    '  -  '  -     '
2nd:    -  '    -   -   '    '  -  '  -     '
3rd:    -  '    -   '   '    '  -  '  -     '
4th:    -  '    -   -   -    '  -  '  -     '
```

The first variant is the meter most likely intended by the author. The second line represents the mentioned alternative scansion. The third and fourth lines show the output of the software tools Scandroid (Hartman 2005) and ZeuScansion, respectively.

Sometimes a line's analysis can be different from the expected one. In fact, well-known poems usually include some metrical variation; this is a stylistic device to break monotony and provide elements of surprise and variation to the reader. In the poem *The More Loving One* by W. H. Auden (Auden 1960), the poet varies the meter several times. An interesting case in point is the stanza

Admirer as I think I am
of stars that do not give a damn,
I cannot, now I see them, say
I missed one terribly all day

where the natural flow of the last line is scanned as two iambs and a double iamb.[4] While the poem itself is written in iambic tetrameters, this last line illustrates the author assigning extra emphasis on the final part: *all day*.

In short, evaluating the output of automatic scansion is somewhat complicated by the possibility of various good interpretations. As we shall see below, when evaluating the scansion task, we use a gold standard that addresses this and accepts several possible outputs as valid.

1.2 *The challenges of scansion*

Scansion is, then, the analysis of rhythmic structure in verse. But what makes it difficult? In the following, we discuss some of the immediate obstacles that have to be overcome to provide accurate annotations of rhythm and stress.

[3] Pyrrhic foot: Two unstressed syllables [--].

[4] Double iamb: two unstressed syllables and two stressed syllables [--''].

1.2.1 Lexical stress patterns do not always apply

The primary piece of information necessary for performing metrical scansion is the *lexical stress* of words. While other elements are also important, the inherent lexical stress of a word is indispensable for the task. Consider the first line of Thomas Hardy's *The voice* (Monroe 1917, p. 131):

Woman much missed, how you call to me, call to me

If we were to simply perform scansion by marking the primary, secondary, and unstressed syllables along the line as provided for the individual words in a dictionary,[5] the result would be

```
 ' -   '    `       '    '  '    -  '   '    -  '
Woman much missed, how you call to me, call to me
```

This poem is in fact composed of four quatrains, where each line is written in dactylic tetrameter throughout,[6] which leads to the following analysis for this line.

```
 ' -   -    '       -    -  '    -  -   '    -  -
Woman much missed, how you call to me, call to me
```

As is obvious, we have to know the *prosodic stress* of the line in order to calculate the meter of the poem; simply knowing the lexical stress of each of the words will not suffice. The lexical stress is the relative emphasis inherent to certain syllables in a word, independently of the word's context. The prosodic stress shows the prominence of each of the syllables within a sentence. We address this problem by using a simplified version of some heuristics proposed by Groves (1998). Groves' rules provide a principled method to exclude some lexically stressed syllables from carrying prosodic stress.

1.2.2 Dividing the stress pattern into feet

The prosodic stress location is important, but knowledge of it is still not sufficient to obtain the intended overall meter of a poem. In order to analyze the meter, each line needs to be divided into plausible feet.

[5] We use the symbol `'` to denote primary stress, the symbol `` ` `` to denote secondary stress, and `-` for unstressed syllables.

[6] Dactyl: a stressed syllable followed by two unstressed syllables `['--]`.

A foot represents a grouping of usually one to three syllables. Returning to the above example by Thomas Hardy, we need to somehow be able to determine that the poem's lines are composed of four dactyls, and thus, that its meter is dactylic tetrameter.

In order to produce a good division of lines into feet, we employ a scoring system that takes into account not only the number of matches of the foot in the stress structure of the poem, but also the length of the feet proposed.

1.2.3 Dealing with out-of-vocabulary words

Automatic scansion is made considerably more difficult by the presence of out-of-vocabulary words. Although the lexical stress of words is not sufficient for scanning a line of poetry, it is nevertheless necessary. For some words, however, it is not available in standard dictionaries. Let us suppose that we are scanning the following line from Henry Wadsworth Longfellow's poem *The song of Hiawatha* (Longfellow 1855, p. 39):

By the shores of Gitche gumee

Here, most dictionaries would lack entries for either *Gitche* or *gumee*. For such cases, we need an informed method or algorithm for assigning lexical stress to out-of-vocabulary words. The use of rare, made-up, or unknown words is, of course, common in poetry. They appear as a result of atypical spellings, are derived through complex morphological processes, or are just nonce words coined for the occasion (cf. John Lennon's *The faulty Bagnose* or *Jabberwocky* by Lewis Carroll, 1916). Usually, the character names in poems also do not appear in dictionaries, and so their scansion cannot be inferred from such knowledge sources. This problem is exacerbated in older poetry (e.g., *Beowulf*). Failure to correctly indicate primary stress in such unknown words results in a lower accuracy of automatic scansion systems.

In order to reduce the occurrence of this type of error, we use an FST-based system that finds words spelled similarly to the target unknown word, with the assumption that their lexical stress will also be similar. More sophisticated algorithms for this purpose have been developed in Agirrezabal *et al.* (2014); such external resources can easily be embedded in ZeuScansion because of its modular design.

2 THE OUTPUT OF ZEUSCANSION

As many different established systems of scansion exist that often vary in minor details, we have chosen a rather conservative approach, which also lends itself to a fairly mechanical, linguistic rule-based implementation. The system distinguishes three levels of stress, marks each line with a stress pattern, and attempts to analyze the predominant meter used in a poem. The following illustrates the analysis produced by our tool of a stanza from Lewis Carroll's poem *Jabberwocky* (Carroll 1916, p. 181):

```
1 He took his vorpal sword in hand:
2 Long time the manxome foe he sought-
3 So rested he by the Tumtum tree,
4 And stood awhile in thought.

1 - ' - `- ' - '
2 ' ' - `' ' - '
3 ' `- - - - `- '
4 - ' -' - '
```

In addition to this, the system also analyzes the different types of feet that make up the whole poem (discussed in more detail below). ZeuScansion supports most of the common types of foot found in English poetry, including *iambs*, *trochees*, *dactyls*, and *anapests*. Table 1 shows a complete listing of the feet supported by the tool.

Table 1:
Metrical feet used in English poetry supported by ZeuScansion

	Stress pattern	Name
Disyllabic feet	- -	pyrrhus
	- '	iamb
	' -	trochee
	' '	spondee
Trisyllabic feet	- - -	tribrach
	' - -	dactyl
	- ' -	amphibrach
	- - '	anapest
	- ' '	bacchius
	' ' -	antibacchius
	' - '	cretic
	' ' '	molossus

Once we have identified the feet used in the whole poem, we can infer the poem's meter. This includes common meters such as:

- Iambic pentameter: Lines composed of 5 iambs, used by Shakespeare in his *Sonnets* (Shakespeare 2011).
- Dactylic hexameter: Lines composed of 6 dactyls, used by Homer in the *Iliad* (Murray 1925).
- Iambic tetrameter: Lines composed of 4 iambs, used by Robert Frost in *Stopping by Woods on a Snowy Evening* (Frost 1979).

For example, if we provide Shakespeare's *Sonnets* (the whole work) as input, ZeuScansion's global analysis concludes it to be written in *iambic pentameter* (line-by-line output omitted here):

```
Syllable stress _'_'_'_'_'
Meter: Iambic pentameter
```

For Longfellow's *The song of Hiawatha*, the result of the global analysis is:

```
Syllable stress '_'_'_'_
Meter: Trochaic tetrameter
```

3 RELATED WORK

Scansion of English verse has attracted attention from numerous scholars for years. There are several books that provide general introductions to prosody in English poetry, for example, Corn (1997) or Steele (1999).

In Gerber (2013), the author compares two existing approaches to scansion: traditional stress metrics and generative metrics. In developing ZeuScansion, we have followed the traditional approach.

A number of projects also attempt to automate the scansion of English verse. Below, we give an overview of some of the current ones.

Logan (1988) documents a set of programs to analyze sound and meter in poetry. This work falls in a general genre of techniques that attempt to analyze the phonological structure of poems following the generative phonological theory outlined by Chomsky and Halle (1968) and described by Brogan (1981).

Scandroid is a program that scans English verse written in either iambic or anapestic meter, designed by Charles O. Hartman (1996;

2005). The source code is publicly available.[7] The program can analyze poems and check if the predominant stress pattern is iambic or anapestic. However, if the input poem's meter is not one of those two, the system forces each line into one of them.

AnalysePoems is another tool for automatic scansion and identification of metrical patterns written by Marc Plamondon (2006). In contrast to Scandroid, AnalysePoems only identifies patterns; it does not impose them. The program also checks the rhyme scheme found in the input poem. It is reportedly developed in *Visual Basic* and the .NET framework; however, neither the program nor the code appear to be available.

Calliope is a similar tool, built on top of Scandroid by Garrett McAleese (2007). It is an attempt to take advantage of linguistic theories of stress assignment in scansion. The program does not seem to be freely available.

Of the current efforts, Greene *et al.* (2010) appears to be the only one that uses statistical methods in the analysis of poetry. For the learning process, *The Sonnets* by Shakespeare was used, as well as a number of other works freely available online.[8] Weighted finite-state transducers were used for stress assignment. As with the other documented projects, we have not obtained an implementation to review.

4 CORPORA

Several different corpora were used for the development of ZeuScansion. These include the pronunciation dictionaries NETtalk (Sejnowski and Rosenberg 1987) and CMU (Weide 1998), which both list pronunciations of words, the number of syllables they contain, as well as indications of primary and secondary stress location. Each employs a slightly different notation, but they are in general quite similar in content as they both mark three levels of stress and show pronunciations:

```
NETTALK format:
@bdIkeS|n     `_'_     S4     abdication     0     (N)

CMU format:
INSPIRATION    IH2 N S P ER0 EY1 SH AH0 N
```

[7] http://oak.conncoll.edu/cohar/Programs.htm
[8] http://www.sonnets.org

We also use a human-annotated poetry corpus obtained from an interactive learning environment program for training people to scan traditionally metered English poetry called For Better For Verse (Tucker 2011).[9] The poems on the site are marked up with TEI P5 coding, a convenient format for poetry markup.[10] The collection of poems is rather homogeneous, the predominant meter of the poems being iambic (92.7% of the lines). The remaining 7.3% lines use trochaic (3.65%), anapestic (2.09%) or dactylic (1.56%) meters. We employ this corpus in order to evaluate the performance of ZeuScansion.

In addition to this source, we downloaded several poems from Project Gutenberg (Hart 1971) for evaluation and testing purposes.[11]

Finally, we used the Wall Street Journal section of the Penn Treebank (Marcus *et al.* 1993) to train a part-of-speech-tagger, the role of which is described below.

5 METHOD

Our tool is constructed around a number of guidelines for scansion developed by Peter L. Groves (1998). It consists of three main components:

(a) an implementation of Groves' rules of scansion – mainly a collection of POS-based stress-assignment rules,

(b) a pronunciation lexicon together with an out-of-vocabulary word-stress guesser, and

(c) a 'plausible foot division' system.

5.1 *Groves' rules*

Groves' rules try to assign stress levels in a way that turns this task, as far as possible, into an objective process driven by lexicon and syntax, independent of more elusive concepts of the poem such as meaning and intent. The rules assign stress as follows:

1. Primarily stressed syllables of content words (nouns, verbs, adjectives, and adverbs) receive **primary stress**.

[9] `http://prosody.lib.virginia.edu`

[10] `http://www.tei-c.org/release/doc/tei-p5-doc/en/html/VE.html`

[11] `http://www.gutenberg.org`

2. Secondarily stressed syllables in polysyllabic content words, primarily stressed syllables in polysyllabic function words (auxiliaries, conjunctions, pronouns, and prepositions) and secondarily stressed syllables in compound words get **secondary stress**.
3. Unstressed syllables of polysyllabic words and monosyllabic function words are **unstressed**.

In Section 6 we present a more elaborate example to illustrate how Groves' rules are implemented.

5.2 *Pronunciation lexicon and out-of-vocabulary word-stress guesser*

To calculate the basic stress pattern of words necessary for Groves' rules, we mainly use the dictionaries mentioned earlier: the CMU pronunciation dictionary and NETtalk. The system first attempts to locate the stress pattern in the smaller NETtalk dictionary (20,000 words) and then falls back to using CMU (125,000 words) if the word is missing in NETtalk. The merged lexicon, where NETtalk pronunciations are given priority, contains about 133,000 words.

In the event that a word cannot be found in either the NETtalk lexicon or the CMU dictionary, we try to guess the stress pattern of the word using an FST-based system, which relies on the hypothesis that similarly spelled words have the same stress pattern.

5.3 *Foot division system*

The final subtask – no less important than the previous ones – is to divide a line's stress pattern into feet, for which we use a scoring system. The scoring system takes two features into account: the number of matches that each possible foot has in the line and the number of syllables that that foot has. More details are given below.

6 ZEUSCANSION: TECHNICAL DETAILS

The structure of the system is divided into the subtasks shown in Figure 1. We begin with preprocessing and tokenization, followed by part-of-speech tagging. Then, we find the lexical stress pattern for each word, guessing the stress patterns for any words not found in the dictionary. After these preliminaries, we apply Groves' scansion rules to determine the prosodic stress and perform some cleanup of the result.

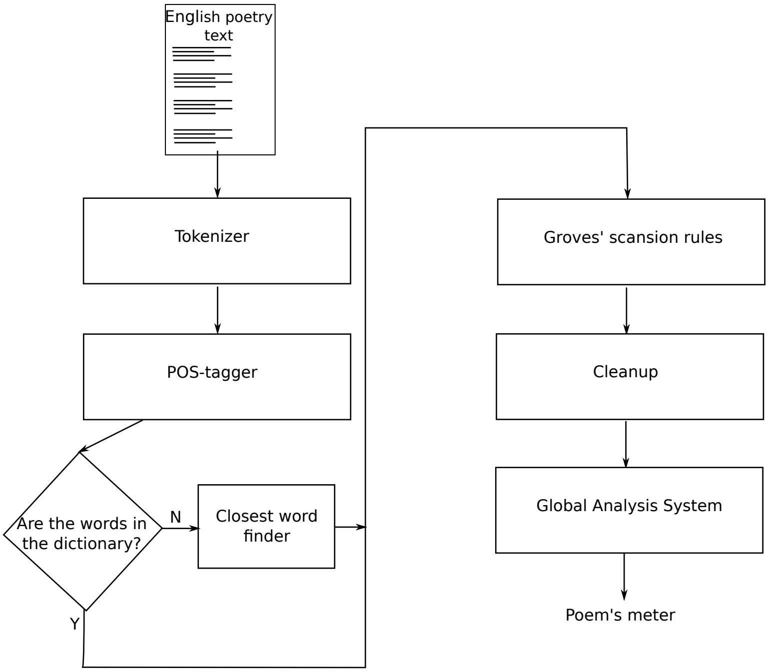


Figure 1: Structure of ZeuScansion

Finally, we calculate the average line stress pattern, which we later try to divide into feet.

The toolchain is implemented as a chain of finite-state transducers, each of them written using the *foma* toolkit (Hulden 2009),[12] save for the part-of-speech tagger, which is a Hidden Markov Model (HMM) implementation (Halácsy *et al.* 2007). We use *Perl* as a glue language to communicate between the components.

6.1 *Preparation of the input data*

After tokenization,[13] we obtain the part-of-speech (POS) tags of the words of the poem. For the POS tagger, we trained Hunpos[14] (Halácsy *et al.* 2007) on the Wall Street Journal section of the Penn Treebank (Marcus *et al.* 1993). While other, more general, corpora might be more suitable for this task, we only need to distinguish between func-

[12] `https://foma.googlecode.com`
[13] Code available at `https://code.google.com/p/foma/wiki/FAQ`.
[14] `https://hunpos.googlecode.com`

tion and non-function words, and thus performance differences would most likely be slight between tagger implementations.

Once the first process is completed, the system starts applying Groves' rules. This process is also encoded as finite-state transducers. To apply the rules, however, we must know the stress pattern of each word. Here, as mentioned above, we resort to a heuristic for assigning lexical stress to out-of-vocabulary words.

The strategy we use to analyze such words is to find a close neighboring word in the dictionary, relying on an intuition that words that differ very little in spelling from the sought-after word are also likely to be pronounced in a similar way, or, at the very least, exhibit the same stress pattern.

6.2 *Finding the closest word*

In order to find what we call the *closest word* in the dictionary, we construct a cascade of finite-state transducers from the existing dictionaries in such a way that, given an input word, it will output the most similar word, according to spelling, using a metric of word distances that we have derived for the purpose. These transducers will perform small specific changes (substitution, insertion, and deletion) on the input word, such as:

- change one vowel,
- change one consonant,
- change two vowels,
- change one vowel and one consonant,
- change two consonants.

Before performing any of these changes, we divide the unknown word into two parts, where the second part represents roughly the last syllable. Then, we perform the aforementioned changes in each part of the word. If, when performing any one of those changes, we find an existing word, the system will return that word and not proceed with the other changes. For example, in the following line from Shakespeare's *Romeo and Juliet* (Shakespeare 1806, p. 77)

And ***usest*** *none in that true use indeed*

we find the word *usest* (the archaic second person singular, simple present form of the verb *use*), which does not appear in our lexicon.

Our closest-word finder begins with the word splitter, which would return `u|sest`. Then, it maps this word to all possible words produced by changing just one vowel in the first part of the word, one vowel in the second part, or changing one consonant. In this example case, after performing some of these changes, the system would determine the closest match according to the scheme above to be *wisest*, and assume that its lexical stress matches that of *usest*. This is achieved by changing one vowel and inserting a consonant at the beginning of the word.

These transducers need to be correctly ordered, as an earlier transducer in the cascade will have priority over later ones. In our cascade, the dictionaries are also included as the very first mapping. If the word is not found in the dictionary, subsequent transducers perform the various mappings, filtering their outputs in such a way as to be constrained against possible words in the dictionary. The actual order in the cascade was determined based on the precision achieved in cross-validation against the NETtalk dictionary. To illustrate this, consider a pair of transducers, one performing just one vowel change and the other changing only one consonant. If the first transducer can guess the correct word stress in, say, 90% of the cases and the other one in 10% of the cases, we order the vowel transducer first in the cascade, and the consonant transducer second. In the case that a close word is not found making the possible mentioned changes, the finder will return the symbol ? as a result.

6.3 *Implementation of Groves' rules*

Once we have obtained the lexical stress for each word, we employ a finite-state transducer that encodes each step in Groves' rules in replacement rules (Beesley and Karttunen 2003).

Groves' rules dictate that the primarily stressed syllable in content words will maintain primary stress. In polysyllabic function words, the syllable carrying primary lexical stress will be assigned secondary stress. Secondary stresses in polysyllablic content words will maintain secondary stress. All other syllables will be unstressed.

The input for these transducers is a string with the structure `word+POS`. The output is the stress pattern of the word after applying Groves' rules, written as `word+stress+POS`. Let's consider a line from Longfellow's poem *The song of Hiawatha*:

changed them thus ***because*** *they mocked you*

For an analysis of the word *because*, the input for the transducer that encodes Groves' rules would be `because+IN`. The lexical resources transducer would locate the word in the dictionary and determine that the second syllable carries primary stress while the first syllable is unstressed. After applying the prosodic stress rules, the system would return that the second syllable should receive secondary stress (instead of the original primary) as the input word is a polysyllabic function word. Hence, the output of the transducer would in this case be `because+-`+IN`.

The last step is to remove all the material not strictly required for working with stress patterns. For the cleanup process, we use a transducer that removes everything before the first + character and everything after the second + character. It then removes all the + characters, so that the only result we get is the bare stress structure of the input word:

```
because+-`+IN → -`
```

6.4 Global analysis

After the stress rules have been applied and we know the stress levels of each syllable of each line, we move to the meter inference process. To this end, we calculate the entire poem's average stress structure. This is encoded by a vector of syllable positions. Each line is examined and for each syllable and its position we add numerical values depending on the syllable's stress. The pseudocode of the average stress calculator is as follows:

```
vector[1..nsylls]=0
foreach line (1..nlines) {
   foreach syllable (1..nsylls) {
      if stress(syllable) == '
          vector[syllable] = vector[syllable] + 2
      if stress(syllable) == `
          vector[syllable] = vector[syllable] + 1
   }
}
```

We illustrate the process with the following excerpt from *The song of Hiawatha* as the input (Longfellow 1855, p. 146):

Barred with streaks of red and yellow$_{1}$
Streaks of blue and bright vermilion$_{2}$
Shone the face of Pau-Puk-Keewis$_{3}$
From his forehead fell his tresses$_{4}$
Smooth and parted like a woman's$_{5}$
Shining bright with oil and plaited$_{6}$
Hung with braids of scented grasses$_{7}$
As among the guests assembled$_{8}$
To the sound of flutes and singing$_{9}$
To the sound of drums and voices$_{10}$
Rose the handsome Pau-Puk-Keewis$_{11}$
And began his mystic dances$_{12}$

According to Groves' rules, the stress values for each line are:

```
'-`-'-'` 1
`-'-'-'- 2
'-'-? 3
--'`'-`- 4
'-`---`- 5
`-'-'-`- 6
'-`-`-`- 7
'-`-`-`- 8
--'-`-`- 9
--'-`-`- 10
'-'-? 11
--`-'-`- 12
```

Our algorithm would then calculate the following:

Syllable	1	2	3	4	5	6	7	8
Σ	14	0	19	1	14	0	12	1
Normalized	0.74	0	1	0.05	0.74	0	0.63	0.05
Stress	'	-	'	-	'	-	'	-

These numbers represent each syllable's average stress over the entire poem. In Figures 2 and 3 we show a graphical representation

Figure 2:
Average stress level per syllable position in Shakespeare's *Sonnets*

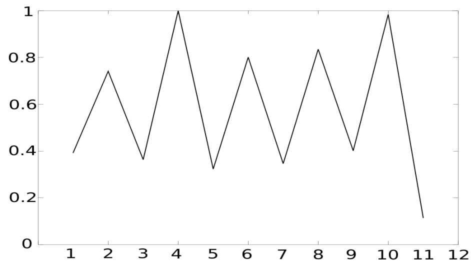
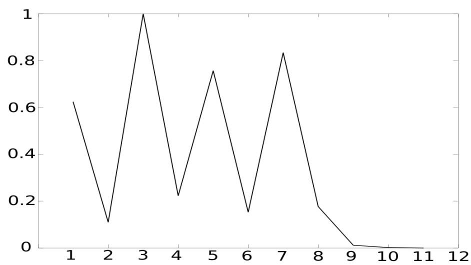


Figure 3:
Average stress level per syllable position in Longfellow's *The song of Hiawatha*



of these numbers based on an analysis of Shakespeare's *Sonnets* and Longfellow's *The song of Hiawatha*. We use 0.5 as a cutoff value: if the normalized average stress for a syllable is greater than this, it is assigned the label *stressed* and otherwise *unstressed*. We assume that all the lines contain the same number of syllables. This naturally leads to difficulties with works with differing syllable counts per line (such as *Phantasmagoria* and other poems by Lewis Carroll, 1869). We set aside the interesting problems surrounding proper normalization and treatment of mixed-line poems for future work.

After the above steps, we attempt to divide the average stress pattern into feet with the goal of producing a global analysis of the poem. In our previous example, it is obvious that the optimal meter to assign is trochaic tetrameter, a sequence of four trochees, but in other cases foot-division can be ambiguous. Consider, for instance, the meter:

- ' - - ' - - ' - - ' -

which could be analyzed as consisting mainly of (1) amphibrachs [- ' -], (2) trochees [' -] and (3) iambs [- ']. All three patterns appear four times in the line. For such cases, we have elaborated a scoring system for selecting the appropriate pattern: we give a weight of 1.0

for hypothetical disyllabic patterns, and a weight of 1.5 for trisyllabic ones. In this example, this would yield the judgement that the structure is amphibrachic tetrameter (1.5×4 matches $= 6$). This example is illustrated in Table 2.

Table 2:
Hypothetical feet for the meter
`-'--'--'--'-`

Foot	Pattern	N¬∫ matches	Score
Amphibrach	`-'-`	4	6
Iamb	`-'`	4	4
Trochee	`'-`	4	4
Anapest	`'--`	3	4.5
Dactyl	`'--`	3	4.5
Pyrrhus	`- -`	3	3

We also attempted to develop an alternative foot-division strategy by taking into account how many syllables were omitted in the analysis. For example, in the previous Longfellow example at line 12, the system would note two unused syllables. The intuition was that a collection of feet that left less unaccounted syllables should be the preferred meter. After evaluating this procedure, however, the results were consistently lower than with the first-mentioned scoring system, which we then chose to use.

7 FURTHER EXPLORATIONS

In the preceding section, we have presented ZeuScansion, an implemented system for scansion, available online.[15] However, we also explored possible improvements for its out-of-vocabulary word-stress guesser. To this end, we developed two alternative approaches based on linguistic generalizations and machine learning techniques. In this section, we will outline how these two systems assign stress to out-of-vocabulary words. While the system described earlier also assigned secondary stresses to words, the alternatives only produce a prediction of the placement of primary stress. However, once the primary stress is assigned, predicting the location of secondary stresses is quite straightforward.

These systems receive a word as input and return the location of primary stress. For example, with the word *introduction* as input, these

[15] `https://github.com/manexagirrezabal/zeuscansion`

guessers should return `--'-`, given that the primary stress is located in the third syllable (*duc*).

Our ultimate goal is to include the best one out of all these approaches in the final ZeuScansion implementation. The source code for these stress assignment tools is made available under the GNU GPL license.[16]

7.1 *Linguistic approach*

For the linguistic approach we have programmed a linguistic toolchain that performs grapheme-to-phoneme conversion (G2P), syllabification and stress assignment.

We first convert the orthographic representation of words to sequences of phonemes, using a G2P system presented in Novak *et al.* (2012).[17] Following this, we syllabify the words using a finite-state syllabification algorithm (Hulden 2006). Our main concern for stress assignment is the weight of the syllables, which might be light or heavy, captured as follows:

- Heavy syllable: The syllable has a coda or ends in a tense vowel.
- Light syllable: Any syllable not classified as heavy.

After this processing, we apply several stress assignment rules that rely on various linguistic generalizations regarding the English vocabulary. The main active rule is the so-called *Latin stress rule* (Halle and Vergnaud 1987), which, despite the name, also applies to many English polysyllabic words. This rule codifies the generalization that heavy syllables tend to attract stress. Below is a description of this, divided into four subrules:

- If the penultimate syllable is light, the antepenultimate syllable is stressed.
- If the penultimate syllable is heavy, it is stressed.
- In the case of disyllabic words, the first syllable is stressed.
- Monosyllabic words are stressed.

Despite the descriptive power of the generalization, examples exist of words where it fails, such as *an|té|nna*, *a|la|bá|ma* or *po|líce*.

[16] `https://github.com/manexagirrezabal/athenarhythm`
[17] `https://github.com/AdolfVonKleist/Phonetisaurus/`

7.2 *Machine learning approach*

In our machine learning approach we have trained a Support Vector Machine (SVM) (Chang and Lin 2011; Fan *et al.* 2008) using the NETTalk stress-annotated dictionary. We treat the stress assignment task as a multi-class classification problem. The class to be assigned is the stress pattern that each word follows, taking into account only the main stress. We extracted 25 different stress patterns from our dictionary, where each stress pattern is a sequence of symbols for stressed and unstressed syllables (`'` and `-`).

We used two different sets of features for the purpose of training the SVMs. In the first set, FS1, we used character bigrams as features, including word boundaries as a special character. In the second feature set, FS2, we used character trigram frequencies, also known as *Wickelfeatures* (Rumelhart and McClelland 1985). For example, given the word *reference*, with FS1 we would train the SVM with the information that the bigrams {#r}, {ef}, {fe}, {er}, {en}, {nc}, {ce}, {e#} appeared once, the bigram {re} twice and all other possible bigrams zero times. These, together with the length of the word, are the training features for the first set. In the second feature set, we include the frequencies of trigrams, in this case {#re}, {ref}, {efe}, {fer}, {ere}, {ren}, {enc}, {nce}, {ce#}. For the example word *reference*, the correct class would be `'-`, indicating that the first syllable carries primary stress.

Naturally, these features need to be encoded as numbers; a simple mapping function performs this mapping. After this, we produced a corpus of 19,528 instances, one instance per word in the dictionary. In the case of FS1, each word was represented using 899 feature–value pairs, while in FS2, 5,495 feature–value pairs were required.

The feature set that yielded the highest performance using cross-validation over the training set was the set consisting of character bigrams and their frequencies (FS1). We trained different support vector machines with varying parameters. The best performing one was a Support Vector Classifier using an RBF/Gaussian kernel, whose parameters C and γ (soft-margin penalty and the Gaussian dispersion) were tuned by a grid-search.

8 EVALUATION

As the gold standard material for evaluation, we used the corpus of scanned poetry For Better For Verse, made available by the University of Virginia, from which we extracted the reference analyses. Sometimes several analyses are given as correct. The results of the evaluation are given in Table 3. 86.78% of syllables are scanned correctly in the best configuration of ZeuScansion. This is slightly below the performance of Scandroid per syllable. As our test corpus is mainly iambic, Scandroid of course has an advantage in that it is fixed to only handle iambic or anapestic feet.

Table 3: ZeuScansion evaluation results against the For Better For Verse corpus

	Scanned lines	Correctly scanned	Accuracy
ZeuScansion	759	199	26.21%
Scandroid	759	326	42.95%

	Scanned sylls.	Correctly scanned	Accuracy
ZeuScansion	7076	5999	86.78%
Scandroid	7076	6353	89.78%

We evaluate our system by checking the error rate obtained by using Levenshtein distance comparing ZeuScansion's output for each line of the analyzed poem against the gold standard scansion. We do this in order not to penalize missing or superfluous syllables, which are sometimes present, with more than 1 count. For example, this line from Longfellow's poem *The song of Hiawatha*,

sent the wildgoose wawa northward

written in trochaic tetrameter, should be scanned as

```
'-'-'-'-
```

while our tool marks the line in question as

```
'-?'-'-
```

after conversion to using only two levels of stress from the original three-level marking. For the conversion, we consider primarily and secondarily stressed syllables stressed, and unstressed syllables unstressed. With the Levenshtein metric we evaluate the distance between the analysis proposed by our tool, ZeuScansion, and the gold

Poem	Correctly classified
The song of Hiawatha	32.03%[18]
Shakespeare's Sonnets	70.13%

Table 4:
Evaluation of the global analysis system (only ZeuScansion)

standard. Obviously, any proposed analysis identical to the gold standard will be assigned a distance of zero. The value that we obtain from using this distance metric can be interpreted as a minimum number of errors in the analysis. In the example, ZeuScansion fails to assign the correct stress pattern to *wildgoose*, because the word does not appear in dictionaries and no similarly spelled word can be found. The minimum Levenshtein distance between the analysis and the reference is two, since changing the third `?` to a `'` and adding a `-` to the analysis would produce the stress pattern given for this line in the gold standard.

We also evaluated the global analysis system using two different works of poetry. The first one is Longfellow's *The song of Hiawatha* and the second one Shakespeare's *Sonnets*. We analyzed the meter of each sonnet in Shakespeare's writing (154 sonnets); in the case of Longfellow's poem we analyzed each stanza (637 stanzas) separately. Shakespeare's sonnets are written in iambic pentameter and *The song of Hiawatha* in trochaic tetrameter. Table 4 reports the accuracy on this task.

8.1 *Out-of-vocabulary word-stress guesser*

Since the out-of-vocabulary word-stress guesser impacts on the overall quality of the system, we have evaluated that component separately. ZeuScansion only uses the similarity approach for the out-of-vocabulary word-stress guessing process. However, we intend to include the linguistic and machine learning approaches in the future as they achieve better results.

The NETtalk pronunciation dictionary was used for evaluating this phase. As some of the methods for stress assignment are data-driven and others not, we evaluated them slightly differently. Both the similarity approach and machine learning approach were evaluated using 10-fold cross-validation. The linguistic approach, however, was evaluated against the whole corpus without any splitting, as it does

[18] 44.58% were classified as amphibraic dimeter.

Table 5:
Evaluation results for the out-of-vocabulary word-stress guesser

	Accuracy
FST-based approach	67.77%
Linguistic approach	**73.62%**
Machine Learning approach	70.98%

not rely on any training data and is essentially an expert system. The results are shown in Table 5.

The highest accuracy is achieved by the linguistic generalization; however, both the results for using SVMs and those for using hand-encoded generalizations are sufficiently close to warrant further research in the improvement of both.

9 DISCUSSION AND FUTURE WORK

In this article, we have presented a basic system for scansion of English poetry. The evaluation results are promising: a qualitative analysis of the remaining errors reveals that the system, while still containing errors vis-à-vis human expert judgements, makes very few egregious errors. We expect to develop the system further in several respects.

We intend to apply new stress-guessing algorithms in ZeuScansion that yield better results. We believe that the general results of the system will improve slightly.

We also plan to add statistical information about the global properties of poems to resolve uncertain cases in a manner consistent with the overall structure of a given poem. Such additions could resolve ambiguous lines and try to make them fit the global pattern of a poem. What we have in mind is the replacement of the part-of-speech tagging process by a deterministic FST-based tagger such as Brill's tagger (Roche and Schabes 1995). This would allow the representation of the entire tool as a single finite-state transducer composed of several subparts. Under such a model, however, we would not be able to use other word-stress guessing algorithms than the similarity approach. In the short term, we also expect to tackle improvements regarding the possibility of analyzing mixed-length lines.

We believe that the availability of a gold-standard corpus of expert scansion offers a valuable improvement in the quantitative assessment of the performance of future systems and modifications.

As noted in Agirrezabal *et al.* (2014), there is still room for improvement in the out-of-vocabulary word-stress allocation systems. One of the main issues is the addition of information about the part of speech to the learning corpus. This is necessary because disyllabic words, which are quite frequent, tend to behave differently along the lines of noun–verb distinction.[19] We believe that with this improvement the accuracy of the linguistic and the machine learning paradigm might see significant gains in accuracy.

To conclude, as our main research project involves both poetry analysis and generation, we intend to use this implementation in the generation of poetry using morphosyntactic patterns following the philosophy of Agirrezabal *et al.* (2013).

ACKNOWLEDGMENTS

The first author's work is funded by a PhD grant from the University of the Basque Country and supported by the Association of the Friends of Bertsolaritza. We are also grateful to the *University of Delaware*, where part of this work was undertaken in the *Department of Linguistics and Cognitive Sciences*. We especially acknowledge the help of professor Jeffrey Heinz. The help of Herbert Tucker, author of the For Better For Verse project, has been fundamental to evaluate our system. We also want to extend thanks to the Scholar's Lab, an arm of the University of Virginia Library, maintainers of For Better For Verse. Joseph Gilbert and Bethany Nowviskie have been most steadily helpful there.

REFERENCES

Manex AGIRREZABAL, Bertol ARRIETA, Aitzol ASTIGARRAGA, and Mans HULDEN (2013), POS-tag based poetry generation with WordNet, *Proceedings of the 2013 European Workshop on Natural Language Generation*, pp. 162–166.

Manex AGIRREZABAL, Jeffrey HEINZ, Mans HULDEN, and Bertol ARRIETA (2014), Assigning stress to out-of-vocabulary words: three approaches, *International Conference on Artificial Intelligence*, 27:105–110.

Wystan H. AUDEN (1960), The more loving one, `https://www.poets.org/poetsorg/poem/more-loving-one`.

[19] If the word is a noun, the stress goes on the first syllable, e.g., *récord*. If it is a verb, the second syllable is stressed, e.g. *recórd*.

Kenneth R. BEESLEY and Lauri KARTTUNEN (2003), Finite-state morphology: Xerox tools and techniques, *CSLI*.

Terry V. F. BROGAN (1981), *English versification, 1570-1980: a reference guide with a global appendix*, Johns Hopkins University Press.

Lewis CARROLL (1869), *Phantasmagoria and Other Poems*, London : Macmillan, `https://archive.org/details/phantasmagoriaot00carrrich`.

Lewis CARROLL (1916), *Alice's Adventures in Wonderland and Through the Looking Glass*, George W. Jacobs & Company, Philadelphia.

Chih-Chung CHANG and Chih-Jen LIN (2011), LIBSVM: A library for support vector machines, *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Noam CHOMSKY and Morris HALLE (1968), *The sound pattern of English*, Harper & Row.

Alfred CORN (1997), *The poem's heartbeat: a manual of prosody*, Copper Canyon Press.

Rong-En FAN, Kai-Wei CHANG, Cho-Jui HSIEH, Xiang-Rui WANG, and Chih-Jen LIN (2008), LIBLINEAR: A library for large linear classification, *The Journal of Machine Learning Research*, 9:1871–1874, software available at `http://www.csie.ntu.edu.tw/~cjlin/liblinear`.

Robert FROST (1979), *The poetry of Robert Frost: the collected poems, complete and unabridged*, Henry Holt and Company.

Paul FUSSELL (1965), *Poetic meter and poetic form*, McGraw Hill.

Natalie GERBER (2013), Stress-based metrics revisited: a comparative exercise in scansion systems and their implications for iambic pentameter, *Thinking Verse*, III:131–168.

Erica GREENE, Tugba BODRUMLU, and Kevin KNIGHT (2010), Automatic analysis of rhythmic poetry with applications to generation and translation, in *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pp. 524–533.

Peter L. GROVES (1998), *Strange music: the metre of the English heroic line*, volume 74 of *ELS Monograph Series*, English Literary Studies, University of Victoria.

Péter HALÁCSY, András KORNAI, and Csaba ORAVECZ (2007), HunPos: an open source trigram tagger, in *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (Interactive Poster and Demonstration Sessions)*, pp. 209–212.

Morris HALLE and Jean-Roger VERGNAUD (1987), *An essay on stress*, MIT Press.

Michael HART (1971), Project Gutenberg, `https://www.gutenberg.org/`.

Charles O. Hartman (1996), *Virtual muse: experiments in computer poetry*, Wesleyan University Press.

Charles O. Hartman (2005), The Scandroid 1.1, `http://oak.conncoll.edu/cohar/Programs.htm`.

Mans Hulden (2006), Finite-state syllabification, in Anssi Yli-Jyrä, Lauri Karttunen, and Juhani Karhumäki, editors, *Finite-State Methods and Natural Language Processing*, volume 4002 of *Lecture Notes in Computer Science*, pp. 86–96, Springer.

Mans Hulden (2009), Foma: a finite-state compiler and library, in *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics (Demonstrations Session)*, pp. 29–32.

Harry M. Logan (1988), Computer analysis of sound and meter in poetry, *College Literature*, 15(1):19–24.

Henry W. Longfellow (1855), *The Song of Hiawatha*, David Bogue, `https://archive.org/details/songhiawathathe00longrich`.

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

William G. M. McAleese (2007), Improving scansion with syntax: an investigation into the effectiveness of a syntactic analysis of poetry by computer using phonological scansion theory, Technical report, Department of Computing Faculty of Mathematics, Computing and Technology The Open University.

Harriet Monroe (1917), *The new poetry: an anthology*, The Macmillan Company, `https://archive.org/details/newpoetryananth01hendgoog`.

Augustus T. Murray (1925), *Homer: The Iliad*, Heinemann, `https://archive.org/details/iliadmurray01homeuoft`.

Josef Novak, Nobuaki Minematsu, and Keikichi Hirose (2012), WFST-based grapheme-to-phoneme conversion: open source tools for alignment, model-building and decoding, in *Proceedings of the 10th Internaltional Workshop on Finite-State Methods and Natural Language Processing*, pp. 45–49.

Marc R. Plamondon (2006), Virtual verse analysis: analysing patterns in poetry, *Literary and Linguistic Computing*, 21(1):127–141.

Margaret Robertson, editor (2007), *Poems published in 1820 by John Keats*, Project Gutenberg, `http://www.gutenberg.org/ebooks/23684`.

Emmanuel Roche and Yves Schabes (1995), Deterministic part-of-speech tagging with finite-state transducers, *Computational Linguistics*, 21(2):227–253.

David E. Rumelhart and James L. McClelland (1985), On learning the past tenses of English verbs, Technical report, Institute for Cognitive Science, University of California, San Diego.

Terrence J. SEJNOWSKI and Charles R. ROSENBERG (1987), Parallel networks that learn to pronounce English text, *Complex Systems*, 1(1):145–168.

William SHAKESPEARE (1806), *Romeo and Juliet*, John Cawthorn.

William SHAKESPEARE (1904), *The tragedy of Hamlet*, Cambridge University Press.

William SHAKESPEARE (2011), *Shakespeare's sonnets*, Project Gutenberg, `https://archive.org/details/shakespearessonn01041gut`.

Timothy STEELE (1999), *All the fun's in how you say a thing: an explanation of meter and versification*, Ohio University Press.

Wallace STEVENS (1923), *Harmonium*, Academy of American Poets.

Herbert F. TUCKER (2011), Poetic data and the news from poems: a *For Better for Verse* memoir, *Victorian Poetry*, 49(2):267–281.

Robert L. WEIDE (1998), *The CMU pronunciation dictionary, release 0.6*, Carnegie Mellon University.

On regular languages over power sets

Tim Fernando
Trinity College Dublin, Ireland

ABSTRACT

The power set of a finite set is used as the alphabet of a string interpreting a sentence of Monadic Second-Order Logic so that the string can be reduced (in straightforward ways) to the symbols occurring in the sentence. Simple extensions to regular expressions are described matching the succinctness of Monadic Second-Order Logic. A link to Goguen and Burstall's notion of an institution is forged, and applied to conceptions within natural language semantics of time based on change. Various reductions of strings are described, along which models can be miniaturized as strings.

Keywords: regular language, power set, MSO, institution

1 INTRODUCTION

Working with more than one alphabet is established practice in finite-state language processing, attested by the popularity of auxiliary symbols (e.g., Kaplan and Kay 1994; Beesley and Karttunen 2003; Yli-Jyrä and Koskenniemi 2004; Hulden 2009). To avoid choosing an alphabet prematurely, implementations commonly treat the alphabet Σ as a dynamic entity that is left underspecified before the finite automaton is constructed in full.[1] Fixing Σ is not always necessary to determine the language denoted by an expression. This is the case with regular expressions; the expression $\emptyset$ denotes the empty set for any alphabet Σ, and the expression ab denotes the singleton set $\{ab\}$ for any alphabet $\Sigma \supseteq \{a, b\}$. Beyond regular expressions, however, there are expressions that denote different languages given different choices of

[1] I am indebted to an anonymous referee for raising this point.

the alphabet Σ. Consider ab's negation (or complement) $\overline{ab}$, which denotes a language

$$\Sigma^* - \{ab\} \;=\; \{s \in \Sigma^* \mid s \neq ab\}$$

that is regular iff Σ is a finite set. To delay fixing Σ to some finite set is to leave open just what the denotation $\Sigma^* - \{ab\}$ of $\overline{ab}$ is. Relative to an alphabet Σ, a symbol c, understood as a string of length one, belongs to that denotation if and only if $c \in \Sigma$. (Σ contains *any* symbol, including c, in the open alphabet system implemented in Beesley and Karttunen 2003.)

Apart from negations, there are many more extensions to regular expressions describing denotations that vary with the choice of alphabet. Consider the sentences of Monadic Second-Order Logic (MSO), which, under a model-theoretic interpretation against strings, capture the regular languages, by a fundamental theorem due independently to Büchi, Elgot and Trakhtenbrot (e.g., Theorem 3.2.11, page 145 in Grädel 2007; Theorem 7.21, page 124 in Libkin 2010). Leaving the precise details of MSO for Section 2 below, suffice it to say (for now) that occurrences of a string symbol a are encoded in a unary predicate symbol P_a for an MSO-sentence such as $\forall x P_a(x)$, saying a occurs at every string position (satisfied by the string aaa but not by the string ab unless $a = b$). We can check if a string over any finite alphabet Σ (hereafter, a Σ-string) satisfies an MSO-sentence φ, but the computation gets costlier as Σ is enlarged. Surely, however, only the symbols that appear in φ matter in satisfying φ or its negation? To investigate this question, let the ***vocabulary of*** φ be the set

$$voc(\varphi) \;:=\; \{a \mid P_a \text{ occurs in } \varphi\}$$

of subscripts of unary predicate symbols appearing in φ. (For example, $\forall x P_a(x)$'s vocabulary $voc(\forall x P_a(x))$ is $\{a\}$.) Now the question is: can we not reduce satisfaction of φ by a Σ-string to satisfaction of φ by a $voc(\varphi)$-string? A simple form such a reduction might take is a function $f : \Sigma^* \to voc(\varphi)^*$ mapping a Σ-string s to a $voc(\varphi)$-string $f(s)$ that satisfies φ if and only if s does

$$s \models \varphi \iff f(s) \models \varphi. \tag{1}$$

Unfortunately, already for φ equal to $\forall x P_a(x)$ and Σ to $\{a, b\}$, it is clear no such function f can exist; the lefthand side of (1) fails for $s = ab$, whereas the righthand side cannot: $a^n \models \forall x P_a(x)$ for all integers $n \geq 0$. Evidently, $voc(\varphi)^*$ is too small to provide the variation necessary for the reduction (1). Enter $(2^{voc(\varphi)})^*$, where the power set 2^A of a set A is the set of all subsets of A. For any MSO-sentence φ and string $s = \alpha_1 \cdots \alpha_n$ of sets α_i, we intersect s componentwise with $voc(\varphi)$ for the $2^{voc(\varphi)}$-string

$$\rho_{voc(\varphi)}(\alpha_1 \cdots \alpha_n) \;:=\; (\alpha_1 \cap voc(\varphi)) \cdots (\alpha_n \cap voc(\varphi)).$$

Then for any finite set Σ, we let MSO_Σ be the set of MSO-sentences with vocabulary contained in Σ

$$MSO_\Sigma \;:=\; \{\varphi \mid \varphi \text{ is an MSO-sentence and } voc(\varphi) \subseteq \Sigma\}$$

and interpret sentences $\varphi \in MSO_\Sigma$ relative to 2^Σ-strings s using a binary relation $\models_\Sigma$ (defined in Section 2) such that

$$s \models_\Sigma \varphi \iff \rho_{voc(\varphi)}(s) \models_{voc(\varphi)} \varphi. \tag{2}$$

The subscripts Σ and $voc(\varphi)$ on $\models$ in the lefthand and righthand sides of (2) track the reduction effected by $\rho_{voc(\varphi)}$ but could otherwise be dropped, had we not already used $\models$ for the satisfaction relation mentioned in (1). Fixing φ's denotation relative to Σ as the set

$$\mathscr{L}_\Sigma(\varphi) \;=\; \left\{s \in (2^\Sigma)^* \mid s \models_\Sigma \varphi\right\}$$

of 2^Σ-strings that $\models_\Sigma$-satisfy φ, we may conclude from (2) that

(†) whatever finite set Σ we use to fix the denotation of φ, it all comes down to $voc(\varphi)$.

Our argument for (†) via (2) rests on modifying MSO-satisfaction $\models$ as it is usually presented over Σ-strings (e.g., Libkin 2010) to one $\models_\Sigma$ over 2^Σ-strings. Without appealing to (†), which might be made precise some other way, we motivate the step from Σ to 2^Σ in our presentation of MSO-models in Section 2, showing, among other things, how that step clarifies what predication and quantification amount to on strings (essentially, preimages and images under $\rho_{voc(\varphi)}$).

Beyond MSO, the reduction (2) is an instance of a general condition built into an abstract model-theoretic approach to specification and programming based on *institutions* (Goguen and Burstall 1992). We adopt this perspective to generalize (2) in Section 3 from $\rho_{voc(\varphi)}$

to functions on strings of sets, manipulating not only the vocabulary but also the length of strings (yielding, at the limit, infinite strings). At the center of this perspective are declarative methods for specifying sets of strings over different alphabets. We focus on methods, including but not limited to MSO, where the alphabets are power sets 2^{Σ} of finite sets Σ.

A multiplicity of such alphabets is useful in the semantics of tense and aspect to measure time at different bounded granularities Σ, tracking finite sets of unary predicates named in Σ. Consider, for instance, Reichenbach's well-known account based on a reference time R, an event time E and a speech time S (Reichenbach 1947). We can picture various temporal relations between an event and a speech as strings of boxes that may or may not contain E or S. For example, the string $\boxed{E}\boxed{S}$ portrays S after E (much like a film or comic strip), which we can verbalize using the simple past or the present perfect, illustrated by (a) and (b) respectively (where the event with time E is Ed's exhalation).

(a) Ed exhaled.
(b) Ed has exhaled.

To represent the difference between (a) and (b), we bring the reference time R into the picture, expanding $\Sigma = \{E,S\}$ to $\Sigma = \{R,E,S\}$ with

(‡) $\boxed{R,E}\boxed{S}$ for the simple past (a), and
$\boxed{E}\boxed{R,S}$ for the present perfect (b),

where a box is drawn instead of the usual curly braces $\{,\}$ for a set construed as a symbol in a string of sets. The difference brought out in (‡) carries significance for anaphora (e.g., Kamp and Reyle 1993, where R is split many ways) and event structure (including an event's consequent state, in Moens and Steedman 1988). Both strings in (‡) can be constructed from simpler strings representing a Reichenbachian analysis of

(i) tense as a relation between R and S, with $\Sigma = \{R,S\}$ and

$$\boxed{R}\boxed{S} \text{ for the past (a), and } \boxed{R,S} \text{ for the present (b)}$$

and

(ii) aspect as a relation between R and E, with $\Sigma = \{R,E\}$ and

$$\boxed{R,E} \text{ for the simple (a), and } \boxed{E}\boxed{R} \text{ for the perfect (b).}$$

Complicating the picture, there are finer analyses of E into aspectual classes going back to Aristotle, Ryle and Vendler (e.g., Dowty 1979) that call for an expansion of $\Sigma = \{R,E,S\}$ to refine the level of granularity (Fernando 2014). A wide ranging hypothesis that the semantics of tense and aspect is finite-state is defended in Fernando (2015), deploying regular languages over power sets, of the kind described below.

Applications to temporal semantics aside, the reader expecting a discussion of finite-state methods applied to phonology, morphology and/or syntax should be warned that such a discussion has been left for someone competent in such matters to take up elsewhere. The present paper claims neither to be the first nor the last word on regular languages over power sets. Its aim simply is to show how to get a handle on the dependence of certain declarative methods on the choice of a finite set Σ of symbols by stepping up to the power set 2^{Σ} of Σ and reducing a string through some function $\rho_{voc(\varphi)}$ or other. MSO provides an obvious point of departure (Section 2), leading to further declarative methods (Section 3).

2 MSO AND RELATED EXTENSIONS OF REGULAR EXPRESSIONS

It is convenient to fix an infinite set Z of symbols a that can appear in unary predicate symbols P_a, from which sentences of MSO are formed. An MSO-sentence φ can have within it only finitely many unary predicate symbols P_a, allowing us to break MSO up into fragments given by finite subsets Σ of Z (no single one of which encompasses all of MSO). In addition to the P_a's, we assume a binary relation symbol S (for successors), from which we can form, for example, the MSO-sentence

$$\forall x\big(P_a(x) \supset \exists y(S(x,y) \land P_b(y))\big)$$

saying that every a-occurrence is succeeded by a b-occurrence. Formal definitions are given in Subsection 2.1 of a satisfaction relation $\models_{\Sigma}$ between (finite) MSO_{Σ}-models and MSO_{Σ}-sentences, built from MSO_{Σ}-formulas with free variables analyzed by suitable expansions of Σ. These expansions are undone by functions ρ_{Σ} on strings that arguably provide the key to predication and quantification over strings. Indeed, the ρ_{Σ}'s pave an easy route to the regularity of MSO, as we show in Subsection 2.2. The functions can be tweaked for useful extensions in Subsection 2.3 of regular expressions, and declarative methods in Section 3 that, like our presentation of MSO via $\models_\Sigma$, meet abstract requirements from Goguen and Burstall (1992).

In what follows, we write *Fin*(A) for the set of finite subsets of a set A. Often but not always, A is Z.

2.1 *MSO-models, formulas and satisfaction*

We restrict our attention to finite models, defining for any integer $n \geq 0$, $[n]$ to be the set of integers from 1 to n,

$$[n] \ := \ \{1, 2, \ldots, n\}$$

and S_n to be the successor (next) relation from i to $i+1$ for $i \in [n-1]$

$$S_n \ := \ \{(1,2), (2,3), \ldots, (n-1, n)\}\,.$$

Given $\Sigma \in \mathit{Fin}(Z)$, let us agree that an *MSO*$_\Sigma$*-model* M is a tuple

$$\langle [n], S_n, \{[\![P_a]\!]\}_{a\in\Sigma}\rangle$$

for some integer $n \geq 0$,[2] such that for each $a \in \Sigma$, $[\![P_a]\!]$ is a subset of $[n]$ interpreting the unary relation symbol P_a. For $A \subseteq \Sigma$, the *A-reduct of* M is the MSO$_A$-model $\langle [n], S_n, \{[\![P_a]\!]\}_{a\in A}\rangle$, keeping only the interpretations $[\![P_a]\!]$ for $a \in A$.

There is a simple bijection *str* from MSO$_\Sigma$-models to 2^Σ-strings, picturing an MSO$_\Sigma$-model $M = \langle [n], S_n, \{[\![P_a]\!]\}_{a\in\Sigma}\rangle$ as the 2^Σ-string $\mathit{str}(M) = \alpha_1 \cdots \alpha_n$ with

$$\alpha_i \ := \ \{a \in \Sigma \mid i \in [\![P_a]\!]\} \qquad\qquad (\text{for } i \in [n]),$$

which inverts to

$$[\![P_a]\!] \ = \ \{i \in [n] \mid a \in \alpha_i\} \qquad\qquad (\text{for } a \in \Sigma).$$

For example, if $\Sigma = \{a, b\}$ and M is $\left\langle [4], S_4, \{[\![P_c]\!]\}_{c\in\Sigma}\right\rangle$ with $[\![P_a]\!] = \{1,2\}$ and $[\![P_b]\!] = \{1,3\}$, then

$$\mathit{str}(M) \ = \ \boxed{a,b}\boxed{a}\boxed{b}\boxed{}$$

(with α_i boxed, as noted in the introduction, to mark them out as string symbols). Strings of boxes with exactly one $a \in \Sigma$ embed Σ^* into $(2^\Sigma)^*$; let $\iota : \Sigma^* \to (2^\Sigma)^*$ map $a_1 \cdots a_n \in \Sigma^n$ to

$$\iota(a_1 \cdots a_n) \ := \ \boxed{a_1}\cdots\boxed{a_n}.$$

[2] We follow Libkin (2010) in allowing a model to have an empty domain/universe.

An advantage in working with $(2^{\Sigma})^*$ rather than Σ^* is that we can intersect a 2^{Σ}-string $\alpha_1 \cdots \alpha_n$ componentwise with any subset A of Σ for the 2^A-string

$$\rho_A(\alpha_1 \cdots \alpha_n) \ := \ (\alpha_1 \cap A) \cdots (\alpha_n \cap A)$$

(generalizing $\rho_{voc(\varphi)}$ in the introduction). The A-reduct of the MSO_{Σ}-model given by the string $\alpha_1 \cdots \alpha_n$ is represented by $\rho_A(\alpha_1 \cdots \alpha_n)$; i.e., for any MSO_{Σ}-model M and MSO_A-model M',

$$\rho_A(\mathit{str}(M)) = \mathit{str}(M') \iff M' \text{ is the } A\text{-reduct of } M.$$

The difference between an MSO_{Σ}-model M and the string $\mathit{str}(M)$ is so slight that we can confuse M harmlessly with $\mathit{str}(M)$ and refer to a 2^{Σ}-string as an MSO_{Σ}-model.

To form MSO-formulas with free variables, let us fix an infinite set *Var* disjoint from Z, $\mathit{Var} \cap Z = \emptyset$, treating each $x \in \mathit{Var}$ as a first-order variable. Given finite subsets Σ of Z and V of *Var*, we define a *$MSO_{\Sigma,V}$-model* to be a $2^{\Sigma \cup V}$-string in which each $x \in V$ occurs exactly once, and collect these in the set $\mathit{Mod}_V(\Sigma)$

$$\mathit{Mod}_V(\Sigma) \ := \ \left\{ s \in (2^{\Sigma \cup V})^* \mid (\forall x \in V) \ \rho_{\{x\}}(s) \in \boxed{}^* \boxed{x} \boxed{}^* \right\}.$$

We define the set $\mathit{MSO}_{\Sigma,V}$ of MSO_{Σ}-formulas φ with free variables in V by induction, alongside sets $\mathscr{L}_{\Sigma,V}(\varphi)$ of strings in $\mathit{Mod}_V(\Sigma)$ that satisfy φ, determining a satisfaction relation

$$\models_{\Sigma,V} \ \subseteq \ \mathit{Mod}_V(\Sigma) \times \mathit{MSO}_{\Sigma,V}$$

between strings $s \in \mathit{Mod}_V(\Sigma)$ and formulas $\varphi \in \mathit{MSO}_{\Sigma,V}$ according to

$$s \models_{\Sigma,V} \varphi \iff s \in \mathscr{L}_{\Sigma,V}(\varphi).$$

The inductive definition consists of six clauses.

(a) If $\{x,y\} \subseteq V$, then $x = y$ and $S(x,y)$ are in $\mathit{MSO}_{\Sigma,V}$, with $x = y$ satisfied by strings in $\mathit{Mod}_V(\Sigma)$ where x and y occur in the same position

$$\mathscr{L}_{\Sigma,V}(x = y) \ := \ \left\{ s \in \mathit{Mod}_V(\Sigma) \mid \rho_{\{x,y\}}(s) \in \boxed{}^* \boxed{x,y} \boxed{}^* \right\}$$

and $S(x,y)$ satisfied by strings in $\mathit{Mod}_V(\Sigma)$ where x occurs immediately before y

$$\mathscr{L}_{\Sigma,V}(S(x,y)) \ := \ \left\{ s \in \mathit{Mod}_V(\Sigma) \mid \rho_{\{x,y\}}(s) \in \boxed{}^* \boxed{x} \boxed{y} \boxed{}^* \right\}.$$

(b) If $a \in \Sigma$ and $x \in V$, then $P_a(x)$ is in $MSO_{\Sigma,V}$ and is satisfied by strings in $Mod_V(\Sigma)$ where the occurrence of x coincides with one of a

$$\mathscr{L}_{\Sigma,V}(P_a(x)) \\ := \left\{s \in Mod_V(\Sigma) \mid \rho_{\{a,x\}}(s) \in \left\{\boxed{}, \boxed{a}\right\}^* \boxed{a,x} \left\{\boxed{}, \boxed{a}\right\}^*\right\}.$$

(c) If $\varphi \in MSO_{\Sigma,V}$ then so is $\neg\varphi$ with $\neg\varphi$ satisfied by strings in $Mod_V(\Sigma)$ that do not satisfy φ

$$\mathscr{L}_{\Sigma,V}(\neg\varphi) := Mod_V(\Sigma) - \mathscr{L}_{\Sigma,V}(\varphi).$$

(d) If φ and ψ are in $MSO_{\Sigma,V}$ then so is $\varphi \wedge \psi$ with $\varphi \wedge \psi$ satisfied by strings in $Mod_V(\Sigma)$ that satisfy both φ and ψ

$$\mathscr{L}_{\Sigma,V}(\varphi \wedge \psi) := \mathscr{L}_{\Sigma,V}(\varphi) \cap \mathscr{L}_{\Sigma,V}(\psi).$$

For quantification, we must be careful that a variable can be reused, as in

$$P_b(x) \wedge \exists x P_a(x),$$

which is equivalent to $P_b(x) \wedge \exists y P_a(y)$ since $\exists x P_a(x)$ and $\exists y P_a(y)$ are.[3] To cater for reuse of $q \in Var \cup Z$, we define an equivalence relation $\sim_q$ between strings s and s' of sets that differ at most on q, putting

$$s' \sim_q s \iff \hat{\rho}_q(s') = \hat{\rho}_q(s),$$

where the function $\hat{\rho}_q$ removes q from a string $\alpha_1 \cdots \alpha_n$ of sets

$$\hat{\rho}_q(\alpha_1 \cdots \alpha_n) := (\alpha_1 - \{q\}) \cdots (\alpha_n - \{q\}).$$

We can now state the last two clauses of our inductive definition of $MSO_{\Sigma,V}$ and $\mathscr{L}_{\Sigma,V}(\varphi)$.

(e) If $\varphi \in MSO_{\Sigma,V\cup\{x\}}$ then $\exists x\varphi$ is in $MSO_{\Sigma,V}$ with $\exists x\varphi$ satisfied by strings in $Mod_V(\Sigma)$ that are $\sim_x$-equivalent to strings in $Mod_{V\cup\{x\}}(\Sigma)$ satisfying φ :

$$\mathscr{L}_{\Sigma,V}(\exists x\varphi) := \left\{s \in Mod_V(\Sigma) \mid (\exists s' \in \mathscr{L}_{\Sigma,V\cup\{x\}}(\varphi))\ s' \sim_x s\right\},$$

which simplifies in case x is not reused

$$\mathscr{L}_{\Sigma,V}(\exists x\varphi) = \left\{\rho_{\Sigma\cup V}(s) \mid s \in \mathscr{L}_{\Sigma,V\cup\{x\}}(\varphi)\right\} \qquad \text{if } x \notin V.$$

[3] We can always avoid reuse in finite formulas, working with finitely many variables.

(f) If $\varphi \in MSO_{\Sigma\cup\{a\},V}$ then $\exists P_a\varphi$ is in $MSO_{\Sigma,V}$ with $\exists P_a\varphi$ satisfied by strings in $Mod_V(\Sigma)$ that are $\sim_a$-equivalent to strings in $Mod_V(\Sigma\cup\{a\})$ satisfying φ :

$$\mathscr{L}_{\Sigma,V}(\exists P_a\varphi) := \left\{s \in Mod_V(\Sigma) \mid (\exists s' \in \mathscr{L}_{\Sigma\cup\{a\},V}(\varphi))\ s' \sim_a s\right\},$$

which simplifies in case P_a is not reused

$$\mathscr{L}_{\Sigma,V}(\exists P_a\varphi) = \left\{\rho_{\Sigma\cup V}(s) \mid s \in \mathscr{L}_{\Sigma\cup\{a\},V}(\varphi)\right\} \qquad \text{if } a \notin \Sigma.$$

We adopt the usual abbreviations: $\varphi \vee \psi$ for $\neg(\neg\varphi \wedge \neg\psi)$, $\forall x\varphi$ for $\neg\exists x\neg\varphi$, etc. Also, we render second-order quantification $\exists P_a$ as $\exists X$, writing $\exists X\varphi$ for $\exists P_a\varphi_a^X$ where a does not occur in φ, and φ_a^X is φ with P_a replacing every occurrence of X. For example, we can express $x < y$ as $\exists X(X(y) \wedge \neg X(x) \wedge \text{closed}(X))$ where closed(X) abbreviates $\forall x\forall y(X(x) \wedge S(x,y) \supset X(y))$, which we can picture as

$$\mathscr{L}_{\{a\},\emptyset}(\text{closed}(P_a)) = \boxed{}^*\boxed{a}^*$$

for the picture

$$\begin{aligned}
&\mathscr{L}_{\emptyset,\{x,y\}}(\exists P_a(P_a(y) \wedge \neg P_a(x) \wedge \text{closed}(P_a))) \\
&\qquad = \left\{\rho_{\{x,y\}}(s) \mid s \in \mathscr{L}_{\{a\},\{x,y\}}(P_a(y) \wedge \neg P_a(x) \wedge \text{closed}(P_a))\right\} \\
&\qquad = \left\{\rho_{\{x,y\}}(s) \mid s \in \boxed{}^*\boxed{x}^*\boxed{}^*\boxed{a}^*\boxed{a,y}\boxed{a}^*\right\} \\
&\qquad = \boxed{}^*\boxed{x}\boxed{}^*\boxed{y}\boxed{}^*
\end{aligned}$$

of $x < y$.

Next comes the pay-off in interpreting MSO-sentences over not just Z-strings but strings of sets. An easy proof by induction on $\varphi \in MSO_{\Sigma,V}$ establishes

Proposition 1 *Let $\Sigma \in Fin(Z)$ and $V \in Fin(Var)$. Then for all sets $A \subseteq \Sigma$ and $U \subseteq V$,*

$$MSO_{A,U} \subseteq MSO_{\Sigma,V}$$

and for all $\varphi \in MSO_{A,U}$,

$$\mathscr{L}_{\Sigma,V}(\varphi) = \left\{s \in Mod_V(\Sigma) \mid \rho_{A\cup U}(s) \in \mathscr{L}_{A,U}(\varphi)\right\}.$$

To pick out $MSO_{\Sigma,V}$-formulas with *no* free variables, we let $V = \emptyset$ for the set

$$MSO_\Sigma = MSO_{\Sigma,\emptyset}$$

of MSO_Σ-sentences, and write $\models_\Sigma$ for $\models_{\Sigma,\emptyset}$, and $\mathscr{L}_\Sigma(\varphi)$ for $\mathscr{L}_{\Sigma,\emptyset}(\varphi)$ (where $\varphi \in \mathit{MSO}_\Sigma$). An immediate corollary to Proposition 1 is that for all $\varphi \in \mathit{MSO}_\Sigma$ and $s \in \mathit{Mod}_\emptyset(\Sigma) = (2^\Sigma)^*$,

$$s \models_\Sigma \varphi \iff \rho_{voc(\varphi)}(s) \models_{voc(\varphi)} \varphi \tag{2}$$

where $voc(\varphi)$ is the smallest subset A of Z such that $\varphi \in \mathit{MSO}_A$

$$voc(\varphi) \;=\; \bigcap\{A \in \mathit{Fin}(Z) \mid \varphi \in \mathit{MSO}_A\}$$

(sharpening the description of $voc(\varphi)$ in the introduction).

2.2 *Regularity*

For any finite sets A and B, the restriction

$$\rho_A^B \;:=\; \rho_A \cap \left((2^B)^* \times (2^B)^*\right)$$

of ρ_A to $(2^B)^*$ is a regular relation – i.e. computed by a finite-state transducer (with one state, mapping $\alpha \subseteq B$ to $\alpha \cap A$). For the preimage (or inverse image) of a language L under a relation R, we borrow the notation

$$\langle R\rangle\, L \;:=\; \left\{s \mid (\exists s' \in L)\; sRs'\right\}$$

from dynamic logic, instead of $R^{-1}L$ which becomes awkward for long R's. We can then rephrase the definition of $\mathit{Mod}_V(\Sigma)$ as

$$\mathit{Mod}_V(\Sigma) \;=\; \bigcap_{x \in V} \left\langle \rho_{\{x\}}^{\Sigma\cup V} \right\rangle \boxed{}^* \boxed{x} \boxed{}^*. \tag{3}$$

Similarly we have

$$\mathscr{L}_{\Sigma,V}(S(x,y)) \;=\; \mathit{Mod}_V(\Sigma) \cap \left\langle \rho_{\{x,y\}}^{\Sigma\cup V} \right\rangle \boxed{}^* \boxed{x}\boxed{y} \boxed{}^* \quad \text{for } x, y \in V$$

and writing θ_A^B for the inverse of ρ_A^B,

$$\begin{aligned}
\mathscr{L}_{\Sigma,V}(\exists x \varphi) \;&=\; \mathit{Mod}_V(\Sigma) \cap \left\langle \rho_{\Sigma\cup V-\{x\}}^{\Sigma\cup V} \right\rangle \left\langle \theta_{\Sigma\cup V-\{x\}}^{\Sigma\cup V\cup\{x\}} \right\rangle \mathscr{L}_{\Sigma,V\cup\{x\}}(\varphi) \\
&=\; \mathit{Mod}_V(\Sigma) \cap \left\langle \theta_{\Sigma\cup V}^{\Sigma\cup V\cup\{x\}} \right\rangle \mathscr{L}_{\Sigma,V\cup\{x\}}(\varphi) \qquad \text{for } x \notin V.
\end{aligned}$$

As regular languages are closed under intersection, complementation and preimages under regular relations (which are themselves closed under inverses), it follows that

Proposition 2 *For every $\Sigma \in Fin(Z)$, $V \in Fin(Var)$ and $\varphi \in MSO_{\Sigma,V}$, the set $\mathscr{L}_{\Sigma,V}(\varphi)$ of strings in $Mod_V(\Sigma)$ that satisfy φ is a regular language.*

The aforementioned Büchi–Elgot–Trakhtenbrot theorem (BET) sidesteps free variables, making do with $MSO_\Sigma = MSO_{\Sigma,\emptyset}$ and a fragment $\models^\Sigma \subseteq \Sigma^* \times MSO_\Sigma$ of $\models_\Sigma \subseteq (2^\Sigma)^* \times MSO_\Sigma$ given by Σ-strings s and $\varphi \in MSO_\Sigma$ such that

$$s \models^\Sigma \varphi \iff \iota(s) \models_\Sigma \varphi$$

(recalling from Subsection 2.1 that $\iota(a_1 \cdots a_n) = \boxed{a_1} \cdots \boxed{a_n}$ for $a_1 \cdots a_n \in \Sigma^n$). A language $L \subseteq \Sigma^*$ is then characterized by BET as regular iff for some sentence $\varphi \in MSO_\Sigma$,

$$L = \left\{ s \in \Sigma^* \mid s \models^\Sigma \varphi \right\}.$$

There is a sense in which the difference between s and $\iota(s)$ is purely cosmetic; a simple one-state finite-state transducer computes ι. But the MSO_Σ-sentences valid in $\models^\Sigma$ need not be valid in $\models_\Sigma$; take the MSO_Σ-sentence

$$spec(\Sigma) \ := \ \forall x \bigvee_{a \in \Sigma} \Big(P_a(x) \wedge \bigwedge_{a' \in \Sigma - \{a\}} \neg P_{a'}(x) \Big)$$

specifying in every string position x, exactly one symbol a from Σ. BET effectively presupposes $spec(\Sigma)$ to extract from $\varphi \in MSO_\Sigma$ the regular language $\{s \in \Sigma^* \mid \iota(s) \models_\Sigma \varphi\}$ over Σ, rather than the full regular language $\mathscr{L}_\Sigma(\varphi)$ over 2^Σ from Proposition 2. To represent a regular language over 2^Σ, BET provides a sentence *not* in MSO_Σ but in MSO_{2^Σ}, which we can translate into MSO_Σ by replacing every subformula $P_\alpha(x)$ (for $\alpha \subseteq \Sigma$) with the conjunction

$$\bigwedge_{a \in \alpha} P_a(x) \ \wedge \bigwedge_{a' \in \Sigma - \alpha} \neg P_{a'}(x)$$

in $MSO_{\Sigma,\{x\}}$ interpretable by $\models_{\Sigma,V}$.[4] Insofar as computations are carried out on syntactic representations (e.g., MSO-formulas) rather than on semantic models (designed largely as theoretical aids to understanding), the explosion from Σ to 2^Σ is computationally worrying in the syntactic step from MSO_Σ to MSO_{2^Σ} rather than in the semantic enrichment of Σ^* to $(2^\Sigma)^*$.

[4] Conversely, we can translate MSO_Σ to MSO_{2^Σ} by replacing subformulas $P_a(x)$, for $a \in \Sigma$, with the disjunction $\bigvee\{P_\alpha(x) \mid \alpha \subseteq \Sigma \text{ and } a \in \alpha\}$ in $MSO_{2^\Sigma,\{x\}}$.

Underlying Proposition 2 is a recipe from $MSO_{\Sigma,V}$ to the regular expressions

$$\begin{aligned} \mathscr{L}_{\emptyset,\{x,y\}}(x=y) &= \boxed{}^* \boxed{x,y} \boxed{}^* \\ \mathscr{L}_{\emptyset,\{x,y\}}(S(x,y)) &= \boxed{}^* \boxed{x}\boxed{y} \boxed{}^* \\ \mathscr{L}_{\{a\},\{x\}}(P_a(x)) &= \left\{\boxed{}, \boxed{a}\right\}^* \boxed{a,x} \left\{\boxed{}, \boxed{a}\right\}^* \end{aligned}$$

closed under conjunction, complementation and preimages under ρ_A^B and θ_A^B. These extended regular expressions are as succinct as the formulas in $MSO_{\Sigma,V}$ they represent (up to a constant factor). That said, if we take the example of *spec*(Σ), we can simplify the recipe for $\mathscr{L}_\Sigma(\textit{spec}(\Sigma))$ considerably to the image of Σ^* under ι

$$\mathscr{L}_\Sigma(\textit{spec}(\Sigma)) = \left\{\boxed{a} \mid a \in \Sigma\right\}^*$$

linear in the size of Σ (as opposed to *spec*(Σ) with quadratically many occurrences of the variable x). The representability of regular languages by regular expressions in general (i.e., Kleene's theorem) raises the question: what useful finite-state tools does MSO add to the usual regular operations? Apart from intersection and complementation (the usual extensions to regular expressions), one tool that MSO_Σ introduces is the idea of a string as a model, the proper formulation of which blows Σ up to its power set 2^Σ (to represent all finite MSO_Σ-models, whether or not they satisfy *spec*(Σ)). Exploiting that blow up, we can define regular relations such as ρ_A^B under which preimages of regular languages are also regular. We modify the relations ρ_A^B in the next subsection, Subsection 2.3, examining the MSO representation of accepting runs of a finite automaton, which is demonstrably more succinct than any available with regular expressions.

2.3 *Some parts and sorts*

Using sets as symbols provides a ready approach to meronymy (i.e., parts); we drop the subscript A on ρ_A for the non-deterministic relation $\trianglerighteq$ of componentwise inclusion between strings of the same length

$$\alpha_1 \cdots \alpha_n \trianglerighteq \beta_1 \cdots \beta_m \iff n = m \text{ and } \alpha_i \supseteq \beta_i \text{ for } i \in [n]$$

called *subsumption* in Fernando (2004). For example, $s \trianglerighteq \rho_A(s)$ for all strings s of sets. A part of reduced length can be obtained by truncating a string s from the front for a suffix s'

$$s \text{ } \textit{suffix} \text{ } s' \iff (\exists s'')\ s = s''s'$$

or from the back for a prefix s'

$$s \text{ } \textit{prefix} \text{ } s' \iff (\exists s'')\ s = s's''.$$

We can then compose the relations $\trianglerighteq$, *suffix* and *prefix* for a notion $\sqsupseteq$ of *containment*

$$\begin{aligned} s \sqsupseteq s' &\iff (\exists s_1, s_2)\ s \trianglerighteq s_1 \text{ and } s_1 \textit{ suffix } s_2 \text{ and } s_2 \textit{ prefix } s' \\ &\iff (\exists u, v)\ s \trianglerighteq us'v \end{aligned}$$

between strings of possibly different lengths. For every atomic $\mathrm{MSO}_{\Sigma,V}$-formula φ, the satisfaction set $\mathscr{L}_{\Sigma,V}(\varphi)$ consists of the strings in $Mod_V(\Sigma)$ with characteristic $\sqsupseteq$-parts, given as follows.

Proposition 3 *For all disjoint finite sets Σ and V,*

$$\begin{aligned} \mathscr{L}_{\Sigma,V}(x = y) &= Mod_V(\Sigma) \cap \langle \sqsupseteq \rangle \boxed{x, y} && \textit{for } x, y \in V \\ \mathscr{L}_{\Sigma,V}(S(x,y)) &= Mod_V(\Sigma) \cap \langle \sqsupseteq \rangle \boxed{x}\boxed{y} && \textit{for } x, y \in V \\ \mathscr{L}_{\Sigma,V}(P_a(x)) &= Mod_V(\Sigma) \cap \langle \sqsupseteq \rangle \boxed{a, x} && \textit{for } a \in \Sigma, \quad x \in V. \end{aligned}$$

Under Proposition 3, each set $\mathscr{L}_{\Sigma,V}(\varphi)$ is the intersection of $Mod_V(\Sigma)$ with a language $\langle \sqsupseteq \rangle s_\varphi$, where s_φ is a string of length ≤ 2 that pictures φ. The obvious picture of $x < y$ is the set $\boxed{x}\boxed{}^*\boxed{y}$ of arbitrarily long strings

$$\mathscr{L}_{\Sigma,V}(x < y) = Mod_V(\Sigma) \cap \langle \sqsupseteq \rangle \boxed{x}\boxed{}^*\boxed{y} \qquad \text{for } x, y \in V$$

which is nonetheless easier to visualize (if not read) than the $MSO_{\emptyset,\{x,y\}}$-formula

$$\exists X\ \ (X(y) \wedge \neg X(x) \wedge (\forall u, v)(X(u) \wedge S(u,v) \supset X(v)))$$

expressing $x < y$. To compress the language $\boxed{x}\boxed{}^*\boxed{y}$ to the string $\boxed{x}\boxed{y}$, we can replace containment $\sqsupseteq$ by *weak containment*

$$\succeq \ := \ \{(\alpha_1 \cdots \alpha_n, x_1 \cdots x_n) \mid x_i = \epsilon \text{ or } x_i \subseteq \alpha_i \text{ for } i \in [n]\}$$

with deletions (x_i equal to the empty string ϵ) allowed anywhere, not just in the front or back of $\alpha_1 \cdots \alpha_n$ or inside any box α_i. (For example, $\boxed{x,a}\boxed{}^n\boxed{y}\boxed{} \succeq \boxed{x}\boxed{y}$ for all integers $n \geq 0$.) Proposition 3 holds with $\sqsupseteq$ and $S(x,y)$ replaced by $\succeq$ and $x < y$ respectively

$$\begin{array}{rcll} \mathscr{L}_{\Sigma,V}(x=y) & = & \mathit{Mod}_V(\Sigma) \cap \langle\succeq\rangle \boxed{x,y} & \text{for } x,y \in V \\ \mathscr{L}_{\Sigma,V}(x<y) & = & \mathit{Mod}_V(\Sigma) \cap \langle\succeq\rangle \boxed{x}\boxed{y} & \text{for } x,y \in V \\ \mathscr{L}_{\Sigma,V}(P_a(x)) & = & \mathit{Mod}_V(\Sigma) \cap \langle\succeq\rangle \boxed{a,x} & \text{for } a \in \Sigma,\ x \in V. \end{array}$$

Whether the part relation R is $\sqsupseteq$ or $\succeq$,[5] what matters for the regularity of $\mathscr{L}_{\Sigma,V}(\varphi)$ is that the restriction of R to $(2^{\Sigma\cup V})^*$

$$R \ \cap \ ((2^{\Sigma\cup V})^* \times (2^{\Sigma\cup V})^*)$$

is computable by a finite-state transducer (for all finite sets Σ and V). Within $\mathit{Mod}_V(\Sigma)$ are part relations $\rho_{\{x\}}$ (for $x \in V$) revealed by the equation

$$\mathit{Mod}_V(\Sigma) \ = \ \bigcap_{x\in V} \left\langle \rho_{\{x\}}^{\Sigma\cup V} \right\rangle \boxed{}^* \boxed{x} \boxed{}^* . \tag{3}$$

Moving from MSO to finite automata, let us rewrite pairs Σ, V as pairs A, Q of disjoint finite sets A and Q, and define an (A,Q)-*automaton* to be a triple $\mathscr{A} = (\rightarrow_{\mathscr{A}}, F_{\mathscr{A}}, q_{\mathscr{A}})$ consisting of

(i) a set $\rightarrow_{\mathscr{A}}$ of triples in $Q \times A \times Q$ specifying $\mathscr{A}$-transitions (where we write $q \xrightarrow{a}_{\mathscr{A}} q'$ instead of $(q,a,q') \in \rightarrow_{\mathscr{A}}$)

(ii) a set $F_{\mathscr{A}} \subseteq Q$ of $\mathscr{A}$-final states, and

(iii) an $\mathscr{A}$-initial state $q_{\mathscr{A}} \in Q$.

Given an (A,Q)-automaton $\mathscr{A}$, an $\mathscr{A}$-*accepting run* is a string

$$\boxed{a_1,q_1}\boxed{a_2,q_2}\cdots\boxed{a_n,q_n} \ \in \ (2^{A\cup Q})^*$$

such that $q_{\mathscr{A}} \xrightarrow{a_1}_{\mathscr{A}} q_1$ and $q_n \in F_{\mathscr{A}}$ and

$$q_{i-1} \xrightarrow{a_i}_{\mathscr{A}} q_i \ \text{ for } \ 1 < i \leq n$$

[5] For the present purposes, we can take a *part relation* to be any fragment R of $\succeq$ (i.e., whenever sRs', $s \succeq s'$). Thus, ρ_A, *suffix*, *prefix*, $\sqsupseteq$ and $\succeq$ are all part relations.

(where for $n = 0$, the empty string ϵ is an $\mathscr{A}$-accepting run iff $q_{\mathscr{A}} \in F_{\mathscr{A}}$). Let *AccRuns*($\mathscr{A}$) be the set of $\mathscr{A}$-accepting runs. Clearly, for all $s \in A^*$,

$$\mathscr{A} \text{ accepts } s \iff (\exists s' \in \mathit{AccRuns}(\mathscr{A}))\ \iota(s) = \rho_A(s')$$

(recalling $\iota(a_1 \cdots a_n) = \boxed{a_1} \cdots \boxed{a_n}$). That is, $\mathscr{A}$ accepts the language

$$\mathscr{L}(\mathscr{A}) \;=\; \langle \iota_A \rangle \left\langle \theta_A^{A \cup Q} \right\rangle \mathit{AccRuns}(\mathscr{A})$$

(recalling θ_A^B is the inverse of ρ_A^B). As for the set *AccRuns*($\mathscr{A}$) of $\mathscr{A}$-accepting runs, we start by collecting strings of pairs from A and Q in

$$\mathit{Pairs}(A,Q) \;:=\; \bigcup_{n \geq 0} \left\{ \boxed{a_1, q_1} \cdots \boxed{a_n, q_n} \mid a_1 \cdots a_n \in A^n \text{ and } q_1 \cdots q_n \in Q^n \right\}.$$

We refine *Pairs*(A, Q) to *AccRuns*($\mathscr{A}$), taking into account

(i) the set *Init*[$\mathscr{A}$] of strings that start with a pair a, q such that $q_{\mathscr{A}} \overset{a}{\rightsquigarrow}_{\mathscr{A}} q$

$$\mathit{Init}[\mathscr{A}] \;:=\; \langle \mathit{prefix} \rangle \left\{ \boxed{a, q} \mid q_{\mathscr{A}} \overset{a}{\rightsquigarrow}_{\mathscr{A}} q \right\}$$

(ii) the set *Final*[$\mathscr{A}$] of strings ending with an $\mathscr{A}$-final state

$$\mathit{Final}[\mathscr{A}] \;:=\; \langle \trianglerighteq \rangle \langle \mathit{suffix} \rangle \left\{ \boxed{q} \mid q \in F_{\mathscr{A}} \right\}$$

and

(iii) the set *Bad*[$\mathscr{A}$] of strings containing $\boxed{q}\boxed{a, q'}$ for triples (q, a, q') *out*side the set $\rightsquigarrow_{\mathscr{A}}$ of $\mathscr{A}$-transitions

$$\mathit{Bad}[\mathscr{A}] \;:=\; \langle \trianglerighteq \rangle \langle \mathit{suffix} \rangle \langle \mathit{prefix} \rangle \left\{ \boxed{q}\boxed{a, q'} \mid (q, a, q') \in Q \times A \times Q \text{ and not } q \overset{a}{\rightsquigarrow}_{\mathscr{A}} q' \right\}.$$

Note that $\langle R \rangle \langle R' \rangle L = \langle R; R' \rangle L$ for all relations R and R' and sets L, where $R; R'$ is the *relational composition of* R *and* R'

$$R; R' \;:=\; \left\{ (s, s') \mid (\exists s'')\ sRs'' \text{ and } s''R's' \right\}$$

(and containment $\sqsupseteq$ is the relational composition of $\trianglerighteq$, *suffix* and *prefix*).

Proposition 4 *For all disjoint finite sets A and Q, and all (A,Q)-automata $\mathscr{A}$, the set AccRuns($\mathscr{A}$) of $\mathscr{A}$-accepting runs consists of all strings in Pairs(A,Q) that belong to Init$[\mathscr{A}]$ and Final$[\mathscr{A}]$ but not to Bad$[\mathscr{A}]$*

$$AccRuns(\mathscr{A}) \;=\; Pairs(A,Q) \cap Init[\mathscr{A}] \cap Final[\mathscr{A}] - Bad[\mathscr{A}].$$

Note that the language *Pairs*(A,Q) can be formed by defining for any finite sets C and D, the set

$$Spec_D(C) \;:=\; \mathscr{L}_{C\cup D}(spec(C)) \;=\; \left\langle \rho_C^{C\cup D} \right\rangle \left\{ \boxed{c} \mid c \in C \right\}^*$$

of $2^{C\cup D}$-strings with exactly one element of C in each box, making

$$Pairs(A,Q) \;=\; Spec_Q(A) \cap Spec_A(Q).$$

The language $\left\{ \boxed{c} \mid c \in C \right\}$ of ρ_C-parts of strings in $Spec_D(C)$ includes strings of any finite length, whereas all strings $\boxed{a,q}$, $\boxed{q}$ and $\boxed{q}\boxed{a,q'}$ pictured in $Init_{\mathscr{A}}$, $Final_{\mathscr{A}}$ and $Bad_{\mathscr{A}}$ have length ≤ 2. This is one sense in which the constraint *Pairs*(A,Q) is global (wide), while *Init*$[\mathscr{A}] \cap$ *Final*$[\mathscr{A}] -$ *Bad*$[\mathscr{A}]$ is local (narrow). A second sense is that *Pairs*(A,Q) captures accepting runs of all (A,Q)-automata, just as $Mod_V(\Sigma)$ in Proposition 3 captures all $\mathrm{MSO}_{\Sigma,V}$-models. That is, *Pairs*(A,Q) and $Mod_V(\Sigma)$ are general, sortal constraints that provide a context (or background) for more specific constraints to differentiate strings of the same sort; this differentiation is effected in Propositions 4 and 3 by attributes or parts that pick out substrings of length bounded by 2. Table 1 outlines the situation.

Table 1:

	sortal (taxonomic)	differential (meronymic)
Proposition 3	$Mod_V(\Sigma)$	$\langle \sqsupseteq \rangle\, s_\varphi$
Proposition 4	$Pairs(A,Q)$	$Init[\mathscr{A}] \cap Final[\mathscr{A}] - Bad[\mathscr{A}]$
	general	specific (to φ, $\mathscr{A}$)
length of part	unbounded (ρ_A)	bounded (≤ 2)

A further difference between the second and third columns of Table 1 is that whereas the sortal constraints $Mod_V(\Sigma)$ and *Pairs*(A,Q) employ deterministic part relations ρ_A, the differential constraints $\langle \sqsupseteq \rangle\, s_\varphi$ and *Init*$[\mathscr{A}] \cap$ *Final*$[\mathscr{A}] -$ *Bad*$[\mathscr{A}]$ employ non-deterministic relations $\sqsupseteq$, *prefix* and the relational composition $\trianglerighteq$; *suffix*. Although it is clear from Subsection 2.1 that the work done by $\sqsupseteq$, *prefix* and $\trianglerighteq$;*suffix* can be done by ρ_A, non-determinism nevertheless arises when introducing existential quantification through the inverse θ_A^B of ρ_A^B (used for the step from $\mathscr{A}$-accepting runs to the language $\mathscr{L}(\mathscr{A})$ accepted by $\mathscr{A}$). But while $\sqsupseteq$, *prefix* and $\trianglerighteq$;*suffix* search inside a string, θ_A^B searches outside. The search by θ_A^B is bounded only because the set B (that serves as its superscript) is finite (with elements of B not in A amounting to auxiliary symbols).

Non-determinism aside, the relations $\sqsupseteq$, *prefix* and $\trianglerighteq$;*suffix* differ from ρ_A and its inverse in relating strings of different lengths. Indeed, Table 1 arose above from the observation that parts with length ≤ 2 suffice for the constraints in the third column. That said, in the next section, we compress strings deterministically without setting any predetermined bounds (such as 2) on the resulting length, for sorts and parts alike.

3 COMPRESSION AND INSTITUTIONS

Having established through Proposition 1 the reduction

$$s \models_\Sigma \varphi \iff \rho_{voc(\varphi)}(s) \models_{voc(\varphi)} \varphi \tag{2}$$

(for all $\varphi \in MSO_\Sigma$ and $s \in (2^\Sigma)^*$), we proceeded to part relations other than ρ_A in Table 1. The present section calls attention to string functions that can (unlike ρ_A) shorten a string, pointing the equivalence (2) and Table 1 in the direction of institutions (Goguen and Burstall 1992). As the length n of a string determines the domain $[n] = \{1, \ldots, n\}$ of the model encoded by the string, compression alters ontology over and above A-reducts produced by ρ_A.

3.1 *From compression to inverse limits*

We can strip off empty boxes at the front and back of a string s by defining

$$unpad(s) \;:=\; \begin{cases} unpad(s') & \text{if } s = \boxed{}s' \text{ or else } s = s'\boxed{} \\ s & \text{otherwise} \end{cases}$$

so that $unpad(s)$ neither begins nor ends with $\boxed{}$, making

$$\boxed{}^*\boxed{x}\boxed{}^* \;=\; \langle unpad \rangle \boxed{x}.$$

Using *unpad*-preimages, we can eliminate Kleene stars from the right side of

$$Mod_V(\Sigma) \;=\; \bigcap_{x \in V} \left\langle \rho^{\Sigma \cup V}_{\{x\}} \right\rangle \;\boxed{}^{*}\boxed{x}\boxed{}^{*} \tag{3}$$

and from the extended regular expressions from Proposition 3 for the sets $\mathscr{L}_{\Sigma,V}(\varphi)$ of strings satisfying formulas $\varphi \in MSO_{\Sigma,V}$. Regular expressions with complementation instead of Kleene star are known in the literature as *star-free regular expressions*, denoting, by a theorem of McNaughton and Papert, the first-order definable sets (Theorem 7.26, page 127, Libkin 2010). We can formulate a notion of Σ-*extended star-free expressions* matching the regular expressions over 2^{Σ}, but while it is easy enough to introduce the constructs $\langle \sqsupseteq \rangle$ and $\langle unpad \rangle$, we need subsets and supersets of Σ to relativize complementation and define the constructs $\left\langle \rho^{B}_{A} \right\rangle$ and $\left\langle \theta^{B}_{A} \right\rangle$, where θ^{B}_{A} is the inverse of ρ^{B}_{A}. On the positive side, this complication is potentially interesting as it suggests a hierarchy between the star-free regular languages and regular languages over 2^{Σ}. Be that as it may, our present concerns lie elsewhere.

Rather than separating the set *Var* of first-order variables from the set Z of subscripts a on unary predicates P_a, we can formulate the requirement on a symbol a that it occur exactly once in $MSO_{\{a\}}$

$$nom(a) \;:=\; \exists x \forall y (P_a(y) \equiv x = y)$$

characteristic of *nominals* in the sense of Hybrid Logic (e.g., Braüner 2014, or “world variables” in Prior 1967, pages 187–197), with

$$\mathscr{L}_{\{a\}}(nom(a)) \;=\; \langle unpad \rangle \boxed{a}.$$

From $nom(a)$, it is a small step to the condition $interval(a)$ that a occur in a string without gaps, which we can express in $\mathrm{MSO}_{\{a\}}$ as

$$interval(a) \;:=\; \exists x \; P_a(x) \;\wedge\; \neg \exists y \; gap_a(y)$$

where $gap_a(y)$ says a does not occur at position y even though it occurs before and after y

$$gap_a(y) \;:=\; \neg P_a(y) \wedge \exists u \exists v \; (u < y \wedge y < v \wedge P_a(u) \wedge \; P_a(v))$$

so that

$$\mathscr{L}_{\{a\}}(interval(a)) \;=\; \langle unpad \rangle \; \boxed{a}^{+}. \tag{4}$$

We can eliminate $\cdot^{+}$ from the right of (4) by defining a function bc that given a string s, compresses blocks α^n of $n > 1$ consecutive occurrences in s of the same symbol α to a single α, leaving s otherwise unchanged

$$bc(s) \;:=\; \begin{cases} bc(\alpha s') & \text{if } s = \alpha\alpha s' \\ \alpha\, bc(\beta s') & \text{if } s = \alpha\beta s' \text{ with } \alpha \neq \beta \\ s & \text{otherwise} \end{cases}$$

so that $\boxed{a}^{+}$ is $\langle bc\rangle \boxed{a}$. In general, bc outputs only stutter-free strings, where a string $\alpha_1\alpha_2\cdots\alpha_n$ is *stutter-free* if $\alpha_i \neq \alpha_{i+1}$ for i from 1 to $n-1$. Construing boxes in a string as moments of time, we can view bc as implementing "McTaggart's dictum that 'there could be no time if nothing changed"' (Prior 1967, page 85). The restriction of bc to any finite alphabet is computable by a finite-state transducer, as are, for all $\Sigma \in \mathit{Fin}(Z)$ and $A \subseteq \Sigma$, the composition $\rho^{\Sigma}_{A}; bc$ for bc^{Σ}_{A}

$$bc^{\Sigma}_{A}(s) \;:=\; bc\big(\rho^{\Sigma}_{A}(s)\big) \qquad \text{for } s \in (2^{\Sigma})^*$$

and the composition $bc^{\Sigma}_{A}; \mathit{unpad}$ for π^{Σ}_{A}

$$\pi^{\Sigma}_{A}(s) \;:=\; \mathit{unpad}\big(bc^{\Sigma}_{A}(s)\big) \;=\; bc\big(\mathit{unpad}(\rho^{\Sigma}_{A}(s))\big) \qquad \text{for } s \in (2^{\Sigma})^*.$$

For $a \in \Sigma$, the (2^{Σ})-strings in which a is an interval are those that $\pi^{\Sigma}_{\{a\}}$ maps to $\boxed{a}$

$$\mathscr{L}_{\Sigma}(\mathit{interval}(a)) \;=\; \left\langle \pi^{\Sigma}_{\{a\}} \right\rangle \boxed{a}.$$

The functions π^{Σ}_{A} compose nicely

$$\text{whenever } A \subseteq B \subseteq \Sigma, \quad \pi^{\Sigma}_{A} \;=\; \pi^{\Sigma}_{B}; \pi^{B}_{A} \tag{5}$$

from which it follows that

$$\begin{aligned} \mathscr{L}_{\Sigma}\Big(\bigwedge_{a\in A} \mathit{interval}(a)\Big) &= \bigcap_{a\in A} \mathscr{L}_{\Sigma}(\mathit{interval}(a)) \\ &= \bigcap_{a\in A} \left\langle \pi^{\Sigma}_{\{a\}} \right\rangle \boxed{a} \\ &= \left\langle \pi^{\Sigma}_{A} \right\rangle \mathit{Interval}(A) \end{aligned}$$

where $\mathit{Interval}(A)$ is the π^{A}_{A}-image of $\bigcap_{a\in A} \left\langle \pi^{A}_{\{a\}} \right\rangle \boxed{a}$

$$\mathit{Interval}(A) \;:=\; \left\{ \pi^{A}_{A}(s) \mid s \in \bigcap_{a\in A} \left\langle \pi^{A}_{\{a\}} \right\rangle \boxed{a} \right\}.$$

Conflating a string s with the language $\{s\}$, observe that $\mathit{Interval}(\{a\}) = \boxed{a}$. For $a \neq a'$, the set $\mathit{Interval}(\{a,a'\})$ consists of thirteen strings, one per interval relation in Allen (1983), which can be partitioned

$$\mathit{Interval}(\{a,a'\}) \;=\; \mathscr{L}\left(a \bigcirc a'\right) \,\cup\, \mathscr{L}(a \prec a') \,\cup\, \mathscr{L}(a' \prec a)$$

between the nine-element set

$$\mathscr{L}\left(a \bigcirc a'\right) \;:=\; \left\{\boxed{a}, \boxed{a'}, \epsilon\right\}\boxed{a,a'}\left\{\boxed{a}, \boxed{a'}, \epsilon\right\}$$

describing overlap $\bigcirc$ between a and a' insofar as for all $s \in \mathit{Interval}(\Sigma)$ with $a, a' \in \Sigma$,

$$s \models_\Sigma \exists x\,(P_a(x) \wedge P_{a'}(x)) \iff \pi^{\Sigma}_{\{a,a'\}}(s) \in \mathscr{L}\left(a \bigcirc a'\right)$$

and the two-element sets

$$\mathscr{L}(a \prec a') \;:=\; \left\{\boxed{a}\boxed{a'}, \boxed{a}\boxed{}\boxed{a'}\right\}$$
$$\mathscr{L}(a' \prec a) \;:=\; \left\{\boxed{a'}\boxed{a}, \boxed{a'}\boxed{}\boxed{a}\right\}$$

describing complete precedence $\prec$ insofar as for all $s \in \mathit{Interval}(\Sigma)$ with $a, a' \in \Sigma$,

$$s \models_\Sigma \forall x \forall y\big((P_a(x) \wedge P_{a'}(y)) \supset x < y\big) \iff \pi^{\Sigma}_{\{a,a'\}}(s) \in \mathscr{L}(a \prec a')$$

and similarly for $a' \prec a$. Event structures are built around the relations $\bigcirc$ and $\prec$ in Kamp and Reyle (1993) (pages 667–674) to express the Russell-Wiener event-based conception of time, a particular elaboration of McTaggart's dictum mentioned above. The sets $\mathit{Interval}(A)$ above provide representations of finite event structures (Fernando 2011).

Requiring that event structures be finite flies against the popularity of, for instance, the real line $\mathbb{R}$ in temporal semantics (e.g., Kamp and Reyle 1993, page 670). But we can approximate any infinite set Z by its set $\mathit{Fin}(Z)$ of finite subsets, using the inverse system $(\mathit{Interval}(A))_{A \in \mathit{Fin}(Z)}$,

$$\pi_{A,B} : \mathit{Interval}(B) \to \mathit{Interval}(A), \quad s \mapsto \pi^B_A(s) \qquad \text{for } A \subseteq B \in \mathit{Fin}(Z)$$

for the *inverse limit*

$$\{\mathbf{a} : \mathit{Fin}(Z) \to \mathit{Fin}(Z)^* \mid \mathbf{a}(A) = \pi_{A,B}(\mathbf{a}(B)) \text{ whenever } A \subseteq B \in \mathit{Fin}(Z)\}$$

consisting of maps $\mathbf{a} : Fin(Z) \to Fin(Z)^*$ that respect the projections $\pi_{A,B}$. An element of that inverse limit, in case $\mathbb{R} \subseteq Z$, is the map $\mathbf{a}_{\mathbb{R}}$ such that for all $r_1 \cdots r_n \in \mathbb{R}^*$,

$$\mathbf{a}_{\mathbb{R}}(\{r_1, r_2, \ldots, r_n\}) \;=\; \boxed{r_1}\boxed{\,}\boxed{r_2}\boxed{\,}\cdots\boxed{r_n} \qquad \text{for } r_1 < r_2 < \cdots < r_n$$

copying $\mathbb{R}$. Notice that compressing strings via $\pi_{A,B}$ allows us to lengthen the strings in the inverse limit. If we remove the compression *bc* in $\pi_{A,B}$, we are left with the map ρ_A that leaves the ontology intact (insofar as the domain of an MSO-model is given by the string length), whilst restricting the vocabulary (for A-reducts).

3.2 *From inverse systems to institutions*

We have left out from the language $Interval(\{a\}) = \boxed{a}$ the string $\boxed{\,}\boxed{a}$ (among many others) that satisfies $interval(a)$, having built *unpad* into π^A_A. Notice that a is bounded to the left in $\boxed{\,}\boxed{a}$

$$\boxed{\,}\boxed{a} \;\models_{\{a\}}\; \exists x \exists y (S(x,y) \wedge P_a(y) \wedge \neg P_a(x))$$

but not in $\boxed{a}$. The functions π^B_A underlying $Interval(A)$ abstract away information about boundedness, which is fine if we assume intervals are bounded (as in Allen 1983). But what if we wish to study intervals that may or may not be left-bounded? Or, for that matter, strings where a may or may not be an interval? The line we pursue in this subsection harks back to Table 1 at the end of Section 2, encoding presuppositions in the second column (e.g., $Mod_V(\Sigma)$), and assertions in the third column (e.g., $\langle \sqsupseteq \rangle s_\varphi$). For instance, we presuppose a string s is stutter-free (i.e., $s = bc(s)$) and assert that a is an interval in s, to replace $Interval(A)$ by the intersection

$$\underbrace{\{bc(s) \mid s \in (2^A)^*\}}_{\text{presupposition}} \;\cap\; \underbrace{\bigcap \left\{ \left\langle \pi^A_{\{a\}} \right\rangle \boxed{a} \mid a \in A \right\}}_{\text{assertion}}$$

of which $\boxed{\,}\boxed{a}$ and $\boxed{a}$ are members, for $a \in A$. More generally, the idea is to refine the inverse system from the previous subsection to certain concrete instances of institutions (in the sense of Goguen and Burstall 1992) given by suitable functions on strings.

More precisely, let Z be a large set of symbols, and f be a function on $\mathit{Fin}(Z)$-strings (e.g., *bc*). For any finite subset A of Z, let $\mathsf{P}_f(A)$ be the image of $(2^A)^*$ under f

$$\mathsf{P}_f(A) \ := \ \{f(s) \mid s \in (2^A)^*\}$$

and let f_A be the composition $f_A = \rho_A; f$

$$f_A(s) \ := \ f(\rho_A(s)) \qquad \text{for } s \in \mathit{Fin}(Z)^*.$$

Thus, $\mathsf{P}_f(A)$ is the image of $\mathit{Fin}(Z)^*$ under f_A. More importantly, for every pair (B,A) of finite subsets of Z such that $A \subseteq B$, we define the function $\mathsf{P}_f(B,A) : \mathsf{P}_f(B) \to \mathsf{P}_f(A)$ sending $s \in \mathsf{P}_f(B)$ to $f_A(s) \in \mathsf{P}_f(A)$

$$\mathsf{P}_f(B,A)(s) \ := \ f_A(s) \qquad \text{for } s \in \mathsf{P}_f(B).$$

Now, to say P_f is an inverse system over $\mathit{Fin}(Z)$ is to require that for all $A \in \mathit{Fin}(Z)$,

(c1) $\mathsf{P}_f(A,A)$ is the identity function on $\mathsf{P}_f(A)$; i.e.,

$$f_A(f(s)) \ = \ f(s) \qquad \text{for all } s \in (2^A)^*$$

and whenever $A \subseteq B \subseteq C \in \mathit{Fin}(Z)$,

(c2) $\mathsf{P}_f(C,A)$ is the composition $\mathsf{P}_f(C,B); \mathsf{P}_f(B,A)$; i.e.,

$$f_A(f(s)) \ = \ f_A(f_B(f(s))) \qquad \text{for all } s \in (2^C)^*.$$

Functions f validating conditions (c1) and (c2) include the identity function on $\mathit{Fin}(Z)^*$ (in which case f_A is ρ_A), *unpad* and *bc* (see Fernando 2014, where inverse systems P_f are referred to as presheaves). The condition (c2) reduces to the condition

$$\text{whenever } A \subseteq B \subseteq \Sigma, \ \ \pi^{\Sigma}_A = \pi^{\Sigma}_B; \pi^B_A \tag{5}$$

from the previous subsection, for f equal to the composition *bc*; *unpad* (meeting also the requirement (c1)). To capture the entry $\mathit{Mod}_V(\Sigma)$ in the second column and row of Table 1 in terms of P_f, we must treat a first-order variable in V as a symbol $a \in Z$ (as in the previous subsection), and build into f both the uniqueness and existence conditions that $\mathit{nom}(a)$ expresses, for $a \in V$. To ensure that no $a \in V$ occur more than once in a string s, we delete occurrences in s of a after its first, setting for all $\alpha_1 \cdots \alpha_n \in \mathit{Fin}(Z)^*$,

$$u_V(\alpha_1 \cdots \alpha_n) \;:=\; \beta_1 \cdots \beta_n \qquad \text{where } \beta_i \;:=\; \alpha_i - \Big(V \cap \bigcup_{j=1}^{i-1} \alpha_j\Big) \text{ for } i \in [n].$$

To ensure each $a \in V$ occurs at least once in the string, we put V at the very end

$$e_V(s\alpha) \;:=\; s(\alpha \cup V)$$

with $e_V(\epsilon) := V$ for the empty string ϵ. Now, if f is the composition $e^V; u^V$ then

$$Mod_V(\Sigma) \;=\; \mathsf{P}_f(\Sigma \cup V)$$

and (c1) and (c2) hold.

The third column of Table 1 calls for further ingredients. Let us define a *Z-form* to be a function *sen* with domain *Fin*(Z) mapping $A \in$ *Fin*(Z) to a set *sen*(A) such that for all $B \in$ *Fin*(Z),

$$sen(A) \cap sen(B) \;\subseteq\; sen(A \cap B)$$

and

$$sen(A) \subseteq sen(B) \;\text{ whenever } A \subseteq B.$$

Given a Z-form *sen*, we can associate every $\varphi \in \bigcup\{sen(A) \mid A \in Fin(Z)\}$ with the finite subset

$$voc(\varphi) \;=\; \bigcap\{A \in Fin(Z) \mid \varphi \in sen(A)\}$$

of Z such that

$$\varphi \in sen(A) \iff voc(\varphi) \subseteq A$$

for all $A \in$ *Fin*(Z). Next, given a function f on *Fin*$(Z)^*$ and a Z-form *sen*, let us agree that a *(f, sen)-specification* $\mathscr{L}$ is a function with domain *Fin*(Z) mapping $A \in$ *Fin*(Z) to a function $\mathscr{L}_A$ with domain *sen*(A) mapping $\varphi \in sen(A)$ to a set $\mathscr{L}_A(\varphi)$ of strings in $\mathsf{P}_f(A)$. The intuition is that $\mathscr{L}_A(\varphi)$ consists of the strings in $\mathsf{P}_f(A)$ that A-satisfy φ

$$s \in \mathscr{L}_A(\varphi) \iff s\ A\text{-satisfies } \varphi \qquad\qquad (\text{for all } s \in \mathsf{P}_f(A)).$$

Putting the ingredients together, let us define a *(Z, f)-quadriplex* to be a 4-tuple $(Fin(Z), \mathsf{P}_f, sen, \mathscr{L})$ such that

(i) P_f is an inverse system over *Fin*(Z)

(ii) *sen* is a Z-form, and

(iii) $\mathscr{L}$ is a (f, sen)-specification.

Note that once Z and f are fixed, only the third and fourth components *sen* and $\mathscr{L}$ of a (Z,f)-quadriplex $(\mathit{Fin}(Z), \mathsf{P}_f, \mathit{sen}, \mathscr{L})$ may vary. To link up with institutions, as defined in Goguen and Burstall (1992), we view

(i) $\mathit{Fin}(Z)$ as a category with morphisms given by $\subseteq$

(ii) P_f as a contravariant functor from $\mathit{Fin}(Z)$ to the category **Set** of sets and functions, and

(iii) *sen* as a (covariant) functor from $\mathit{Fin}(\Phi)$ to **Set** such that whenever $A \subseteq B \in \mathit{Fin}(Z)$, $\mathit{sen}(A,B)$ is the inclusion $\mathit{sen}(A) \hookrightarrow \mathit{sen}(B)$.

The one remaining condition a (Z,f)-quadriplex must meet to be an institution is that for all $A \subseteq B \in \mathit{Fin}(Z)$ and $\varphi \in \mathit{sen}(A)$,

$$s \in \mathscr{L}_B(\varphi) \iff f_A(s) \in \mathscr{L}_A(\varphi) \qquad \text{(for all } s \in \mathsf{P}_f(B))$$

which we can put as the equation

$$\mathscr{L}_B(\varphi) \;=\; \mathsf{P}_f(B) \cap \langle f_A \rangle \; \mathscr{L}_A(\varphi).$$

In fact, the special case $A = \mathit{voc}(\varphi)$ suffices.

Proposition 5 *Given a set Z and function f on $\mathit{Fin}(Z)^*$, a (Z,f)-quadriplex $(\mathit{Fin}(Z), \mathsf{P}_f, \mathit{sen}, \mathscr{L})$ is an institution iff for all $\Sigma \in \mathit{Fin}(Z)$ and $\varphi \in \mathit{sen}(\Sigma)$,*

$$\mathscr{L}_\Sigma(\varphi) \;=\; \mathsf{P}_f(\Sigma) \cap \left\langle f_{\mathit{voc}(\varphi)} \right\rangle \; \mathscr{L}_{\mathit{voc}(\varphi)}(\varphi) \,. \tag{6}$$

If f is the identity on $\mathit{Fin}(Z)^*$, and $\mathit{sen}(\Sigma)$ is MSO_Σ, then (6) becomes the equivalence

$$s \models_\Sigma \varphi \iff \rho_{\mathit{voc}(\varphi)}(s) \models_{\mathit{voc}(\varphi)} \varphi \tag{2}$$

for all $\varphi \in \mathit{MSO}_\Sigma$ and $s \in (2^\Sigma)^*$. (6) also represents the division in Table 1 between column 2 ($\mathsf{P}_f(\Sigma)$) and column 3 ($\left\langle f_{\mathit{voc}(\varphi)} \right\rangle \mathscr{L}_{\mathit{voc}(\varphi)}(\varphi)$), whilst leaving open the possibility that f is not the identity function on $\mathit{Fin}(Z)^*$ nor is φ an MSO-formula.

Under (6), we may assume without loss of generality that *sen* and $\mathscr{L}$ have the following form. For every $\Sigma \in \mathit{Fin}(Z)$, there is a set $\mathrm{Expr}(\Sigma)$ of expressions e with denotations $[\![e]\!] \subseteq (2^\Sigma)^*$ such that $\mathit{sen}(\Sigma) = 2^\Sigma \times \mathrm{Expr}(\Sigma)$ consists of pairs (A,e) of subsets $A \subseteq \Sigma$ and $e \in \mathrm{Expr}(\Sigma)$ with $\mathit{voc}(A,e) = A$ and

$$\mathscr{L}_\Sigma(A,e) \;=\; \mathsf{P}_f(\Sigma) \cap \langle f_A \rangle \, [\![e]\!]. \tag{7}$$

An instructive example is provided by A equal to $\{a\}$, and e equal to the extended regular expression $\langle\sqsupseteq\rangle\,\boxed{a}\boxed{a}$ or equivalently, the MSO$_{\{a\}}$-sentence

$$\exists x \exists y\ (S(x,y) \wedge P_a(x) \wedge P_a(y)).$$

The righthand side of (7) can never hold with $f = bc$; there is *no* $s \in (2^{\Sigma})^+$ such that $bc_{\{a\}}(s) \sqsupseteq \boxed{a}\boxed{a}$. A slight revision, however, makes the right hand side bc-satisfiable; introduce a symbol $b \neq a$ for A equal to $\{a, b\}$ and e equal to $\langle\sqsupseteq\rangle\,\boxed{a,b}\boxed{a}$ or the MSO$_{\{a,b\}}$-sentence

$$\exists x \exists y\ (S(x,y) \wedge P_a(x) \wedge P_a(y) \wedge P_b(x)).$$

In general, we can neutralize block compression bc on a string s by adding a fresh symbol to alternating boxes in s, which bc then leaves unchanged, since

$$bc(s) = s \iff s \text{ is stutter-free}$$

(recalling that $\alpha_1 \cdots \alpha_n$ is *stutter-free* if $\alpha_i \neq \alpha_{i+1}$ for $1 \leq i < n$). Similarly, we can add negations $\overline{a}$ of symbols a in A through a function cl_A

$$cl_A(\alpha_1 \cdots \alpha_n) \ := \ \beta_1 \cdots \beta_n \ \text{ where } \beta_i := \alpha_i \cup \{\overline{a} \mid a \in A - \alpha_i\} \text{ for } i \in [n]$$

to express bc_A^{Σ} in terms of π_B^{Σ}

$$bc_A^{\Sigma} \ = \ cl_A; \pi_{c(A)}^{\Sigma}; \rho_A \quad \text{where} \ \ c(A) := A \cup \{\overline{a} \mid a \in A\}$$

treating $\overline{a} \in c(A) - A$ as an auxiliary symbol, and

$$bc_A^{\Sigma}; cl_A \ = \ cl_A; \pi_{c(A)}^{\Sigma}.$$

Returning to (7) with $f = bc$, we can say a is bounded to the left

$$\mathscr{L}_{\Sigma}(\{a\}, \exists x(\neg P_a(x) \wedge \forall y(P_a(y) \supset x < y))) \ = \ \left\langle bc_{\{a\}}^{\Sigma} \right\rangle \langle \mathit{prefix} \rangle \ \boxed{}$$

applying *prefix* after bc, and say a overlaps a'

$$\mathscr{L}_{\Sigma}(\{a, a'\}, \exists x(P_a(x) \wedge P_{a'}(x))) \ = \ \left\langle bc_{\{a,a'\}}^{\Sigma} \right\rangle \langle\sqsupseteq\rangle \ \boxed{a, a'}$$

applying containment $\sqsupseteq$ after bc. It is clear that *unpad* is just one of many relations that can come after bc_A^{Σ} (leading, in this case, to $\pi_A^{\Sigma} = bc_A^{\Sigma}; \mathit{unpad}$). The projection ρ_A^{Σ} in $bc_A^{\Sigma} = \rho_A^{\Sigma}; bc$ changes the granularity from Σ to A before bc reduces the ontology to suit A, and part relations (such as *prefix*, containment $\sqsupseteq$ or *unpad*) pick out a temporal span to frame a string (such as $\boxed{}$ or $\boxed{a, a'}$) picturing an assertion (e.g., left-boundeness, overlap). We are dividing here the choice of an expression e_φ denoting the language $\mathscr{L}_{voc(\varphi)}(\varphi)$ in Proposition 5 between a relation R and a string s for $e_\varphi = \langle R \rangle s$. Such a choice presupposes the finite approximability of the model of interest via the inverse limit of P_f (the discreteness of strings mirroring the bounded granularity of natural language statements, rife with talk of "the next moment"). Finite approximability is not only plausible but arguably implicit in accounts such as Reichenbach (1947) of tense and aspect.

4 CONCLUSION

There is no question that as declarative devices specifying sets of strings accepted by finite automata, regular expressions are more popular than MSO. What MSO offers, however, is a model-theoretic perspective on strings with computable notions of entailment (inclusions between regular languages being decidable), in addition to Boolean connectives that expose deficiencies in succinctness of regular expressions (e.g., Gelade and Neven 2012). Mapping a finite automaton $\mathscr{A}$ to a regular expression denoting the language $\mathscr{L}(\mathscr{A})$ accepted by $\mathscr{A}$ can have exponential cost (Ehrenfeucht and Zeiger 1976; Holzer and Kutrib 2010). A more concise representation of $\mathscr{L}(\mathscr{A})$ existentially quantifies away the internal states from the accepting runs of $\mathscr{A}$ (analyzed in Proposition 4 above). Not only can this be carried out in MSO (proving one half of the Büchi–Elgot–Trakhtenbrot theorem), but it is well-known that MSO-sentences can be far more succinct than finite automata (e.g., Libkin 2010, pages 124–125, and 135–136). To match the succinctness of MSO, regular expressions over alphabets 2^Σ (for finite sets Σ) are extended with preimages and images under homomorphisms ρ_A that output A-reducts, for $A \subseteq \Sigma$.

The step from Σ up to 2^Σ is justified by the various notions of part between strings of sets, given by ρ_A, subsumption $\trianglerighteq$, *prefix*, *suffix*, block compression *bc* and *unpad*, all computable (over 2^Σ) by finite-state transducers. Reducts between vocabularies are composed with compression within a fixed vocabulary to fit ontology against the vocabulary. An inverse limit construction (turning compression around to extension) takes us beyond the finite models of MSO to infinite time-lines, approximated at granularity Σ by strings over the alphabet 2^{Σ}. Different finite sets Σ induce different notions $\models_{\Sigma}$ of satisfaction that form institutions, under certain minimal smoothness conditions (used to establish the Büchi–Elgot–Trakhtenbrot theorem in Section 2).

ACKNOWLEDGEMENTS

My thanks to Mark-Jan Nederhof for his editorship and four anonymous journal referees for their comments and help.

REFERENCES

James F. ALLEN (1983), Maintaining knowledge about temporal intervals, *Communications of the ACM*, 26(11):832–843.

Kenneth R. BEESLEY and Lauri KARTTUNEN (2003), *Finite State Morphology*, CSLI Publications, Stanford.

Torben BRAÜNER (2014), Hybrid Logic, The Stanford Encyclopedia of Philosophy, http://plato.stanford.edu/archives/spr2014/entries/logic-hybrid/.

David R. DOWTY (1979), *Word Meaning and Montague Grammar*, Reidel, Dordrecht.

Andrzej EHRENFEUCHT and Paul ZEIGER (1976), Complexity measures for regular expressions, *J. Comput. Syst. Sci.*, 12(2):134–146.

Tim FERNANDO (2004), A finite-state approach to events in natural language semantics, *Journal of Logic and Computation*, 14(1):79–92.

Tim FERNANDO (2011), Finite-state representations embodying temporal relations, in *Proceedings 9th International Workshop on Finite State Methods and Natural Language Processing*, pp. 12–20.

Tim FERNANDO (2014), Incremental semantic scales by strings, in *Proceedings EACL 2014 Workshop on Type Theory and Natural Language Semantics (TTNLS)*, pp. 63–71.

Tim FERNANDO (2015), The semantics of tense and aspect: A finite-state perspective, in S. LAPPIN and C. FOX, editors, *Handbook of Contemporary Semantic Theory*, pp. 203–236, Wiley-Blackwell, second edition.

Wouter GELADE and Frank NEVEN (2012), Succinctness of the complement and negation of regular expressions, *ACM Trans. Comput. Log.*, 13(1):4.1–4.19.

Joseph GOGUEN and Rod BURSTALL (1992), Institutions: Abstract model theory for specification and programming, *J. ACM*, 39(1):95–146.

Erich GRÄDEL (2007), Finite model theory and descriptive complexity, in *Finite Model Theory and Its Applications*, pp. 125–230, Springer.

Markus HOLZER and Martin KUTRIB (2010), The complexity of regular(-like) expressions, in *Developments in Language Theory*, pp. 16–30, Springer.

Mans HULDEN (2009), Regular expressions and predicate logic in finite-state language processing, in *Finite-State Methods and Natural Language Processing*, pp. 82–97, IOS Press.

Hans KAMP and Uwe REYLE (1993), *From Discourse to Logic*, Kluwer Academic Publishers, Dordrecht.

Ronald M. KAPLAN and Martin KAY (1994), Regular models of phonological rule systems, *Computational Linguistics*, 20(3):331–378.

Leonid LIBKIN (2010), *Elements of Finite Model Theory*, Springer.

Marc MOENS and Mark STEEDMAN (1988), Temporal ontology and temporal reference, *Computational Linguistics*, 14(2):15–28.

Arthur N. PRIOR (1967), *Past, Present and Future*, Clarendon Press, Oxford.

Hans REICHENBACH (1947), *Elements of Symbolic Logic*, London, Macmillan.

Anssi YLI-JYRÄ and Kimmo KOSKENNIEMI (2004), Compiling contextual restrictions on strings into finite-state automata, in *Proceedings of the Eindhoven FASTAR Days*.

Data-oriented parsing with discontinuous constituents and function tags

Andreas van Cranenburgh[1,2]*, Remko Scha*[2]*, and Rens Bod*[2]
[1] Huygens ING, Royal Netherlands Academy of Arts and Sciences
[2] Institute for Logic, Language and Computation, University of Amsterdam

ABSTRACT

Statistical parsers are effective but are typically limited to producing projective dependencies or constituents. On the other hand, linguistically rich parsers recognize non-local relations and analyze both form and function phenomena but rely on extensive manual grammar engineering. We combine advantages of the two by building a statistical parser that produces richer analyses.

We investigate new techniques to implement treebank-based parsers that allow for discontinuous constituents. We present two systems. One system is based on a Linear Context-Free Rewriting System (LCFRS), while using a Probabilistic Discontinuous Tree-Substitution Grammar (PDTSG) to improve disambiguation performance. Another system encodes discontinuities in the labels of phrase-structure trees, allowing for efficient context-free grammar parsing.

The two systems demonstrate that tree fragments as used in tree-substitution grammar improve disambiguation performance while capturing non-local relations on an as-needed basis. Additionally, we present results for models that produce function tags, resulting in a more linguistically adequate model of the data. We report substantial accuracy improvements in discontinuous parsing for German, English, and Dutch, including results on spoken Dutch.

Keywords: discontinuous constituents, statistical parsing, tree-substitution grammar

This article is a substantially revised and extended version of van Cranenburgh and Bod (2013). While finishing this article, we learned with great sadness of the passing of our co-author Remko Scha. We dedicate this article to his memory.

1 INTRODUCTION

Probabilistic algorithms for parsing and disambiguation select the most probable analysis for a given sentence in accordance with a certain probability distribution. A fundamental property of such algorithms is thus the definition of the space of *possible sentence structures* that constitutes the domain of the probability distribution. Modern statistical parsers are often automatically derived from corpora of syntactically annotated sentences ("treebanks"). In this case, the "linguistic backbone" of the probabilistic grammar naturally depends on the convention for encoding syntactic structure that was used in annotating the corpus.

When different parsing and disambiguation algorithms are applied to the same treebank, their relative accuracies can be objectively assessed if the treebank is split into a training set (that is used to induce a grammar and its probabilities) and a test set (that provides a "gold standard" to assess the performance of the system). This is common practice now. In many cases, however, the linguistic significance of these evaluations may be questioned, since the test sets consist of phrase-structure trees, i.e., part-whole structures where all parts are contiguous chunks. Non-local syntactic relations are not represented in these trees; utterances in which such relations occur are therefore skipped or incorrectly annotated.

For certain practical applications this restriction may be harmless, but from a linguistic (and cognitive) viewpoint it cannot be defended. Since Chomsky's transformational-generative grammar, there have been many proposals for formal grammars with a less narrow scope. Some of these formalisms have been employed to annotate large corpora; in principle, they can thus be used in treebank grammars extracted from these corpora.

The Penn treebank, for instance, enriches its phrase-structure representations with "empty constituents" that share an index with the constituent that, from a transformational perspective, would be analyzed as originating in that position. Most grammars based on the Penn treebank ignore this information, but it was used by, e.g., Johnson (2002), Dienes and Dubey (2003), and Gabbard *et al.* (2006).

Another perspective on non-local syntactic dependencies generalizes the notion of a "syntactic constituent," in that it allows "discontinuous constituent structures," where a non-terminal node dominates a lexical yield that consists of different non-contiguous parts (McCawley 1982). Several German and Dutch treebanks have been annotated in terms of discontinuous constituency, and some statistical parsers have been developed that use these treebanks. Also, phrase structures with co-indexed traces can be converted into discontinuous constituent structures; the Penn treebank can therefore be transformed and used in the discontinuous constituency approach (Evang and Kallmeyer 2011). Figure 1 shows an example of a tree with discontinuous constituents.

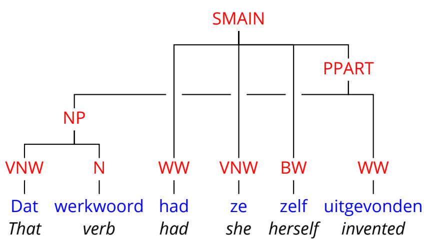
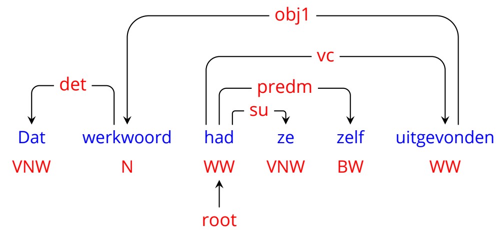


Figure 1:
A tree from the Dutch Alpino treebank (van der Beek *et al.* 2002). PPART is a discontinuous constituent (indicated with crossing branches) due to its extraposed NP object. Part-of-speech tags: VNW = pronoun, N = noun, WW = verb, BW = adverb. The tags also contain additional morphological features not shown here that distinguish personal pronouns from others, auxiliary verbs from main verbs, etc.

It is an annotation choice to employ discontinuous constituents; some treebanks elect not to model non-local phenomena, while others may choose different mechanisms. For example, two German treebanks employ discontinuous constituents (Skut *et al.* 1997; Brants *et al.* 2002), while another German treebank does not (Telljohann *et al.* 2004, 2012). The annotation scheme of the latter treebank lacks information expressed in the former two. For instance, it cannot encode the heads of non-local modifiers; with discontinuous constituents, a modifier is a sibling of its head, regardless of their configuration. On the other hand, the co-indexed traces of the Penn treebank provide more information than discontinuous constituents, because they assume that constituents have been moved from somewhere else in the tree and encode the original position. Discontinuous constituents describe surface structure without making such assumptions. Some phenomena that can be analyzed with discontinuous constituents are extraposition, topicalization, scrambling, and parentheticals; cf. Maier *et al.* (2014) for an overview of such phenomena in German.

Figure 2:
A dependency structure derived from the tree in Figure 1. The *obj1* arc makes this structure non-projective.



The notion of discontinuous constituents in annotation is useful to bridge the gap between the information represented in constituency and dependency structures. Constituency structures capture the hierarchical structure of phrases – which is useful for identifying re-usable elements; discontinuous constituents extend this to allow for arbitrary non-local relations that may arise due to such phenomena as extraposition and free word order. There is a close relation of discontinuous constituency to non-projectivity in dependency structures (Maier and Lichte 2011). Compare Figure 2, which shows a dependency structure for the constituency tree in Figure 1. Note that in this dependency structure, the edge labels are grammatical functions present in the original treebank, while the constituent labels in Figure 1 are syntactic categories. The dependency structure encodes the non-local relations within the discontinuous constituent. On the other hand, it does not represent the hierarchical grouping given by the NP and PPART constituents. By encoding both hierarchical and non-local information, trees with discontinuous constituents combine the advantages of constituency and dependency structures. We will also come back to grammatical function labels.

This paper is concerned with treebank-based parsing algorithms that accept discontinuous constituents. It takes as its point of departure work by Kallmeyer and Maier (2010, 2013) that represents discontinuous structures in terms of a string-rewriting version of Linear Context-Free Rewriting Systems (Section 3.1). In addition, we employ Tree-Substitution Grammar (TSG). We make the following contributions:

1. We discuss the notions of competence and performance in (computational) linguistics (Section 2). We argue that instead of focussing on the search for the formal (competence) grammar with the right capacity for natural language, we can consider performance aspects such as cognitive limitations and pruning strategies.

2. We show that Tree-Substitution Grammar can be applied to discontinuous constituents (Section 3.2) and that it is possible, using a transformation, to parse with a Tree-Substitution Grammar without having to write a separate parser for this formalism (Section 4.2).
3. We induce a tree-substitution grammar from a treebank (Section 5) using a method called Double-DOP (Sangati and Zuidema 2011). This method extracts a set of recurring tree fragments. We show that compared to another method which implicitly works with all possible fragments, this explicit method offers advantages in both accuracy and efficiency (Section 4.2.1, Section 9).
4. Fragments make it possible to treat discontinuous constituency as a statistical phenomenon within an encompassing context-free framework (Section 4.1, Section 7); this yields a considerable efficiency improvement without hurting accuracy (Section 9).
5. Finally, we present an evaluation on three languages. We employ manual state splits from previous work for improved performance (Section 8) and discuss methods and results for grammars that produce function tags in addition to phrasal labels (Section 8.3).

This work explores parsing discontinuous constituents with Linear Context-Free Rewriting Systems and Context-Free Grammar, as well as with and without the use of tree fragments through tree substitution. Figure 3 gives an overview of these systems and how they are combined in a coarse-to-fine pipeline (cf. Section 6.4).

2 THE DIVISION OF LABOR BETWEEN COMPETENCE AND PERFORMANCE

Traditionally, two aspects of language cognition have been distinguished: competence and performance (Chomsky 1965). Linguistic competence comprises a language user's "knowledge of language," usually described as a system of rules, while linguistic performance includes the details of the user's production and comprehension behavior. For a computational model, its syntactic competence defines the set of possible sentences that it can process in principle, and the structures it may assign to them, while its performance includes such

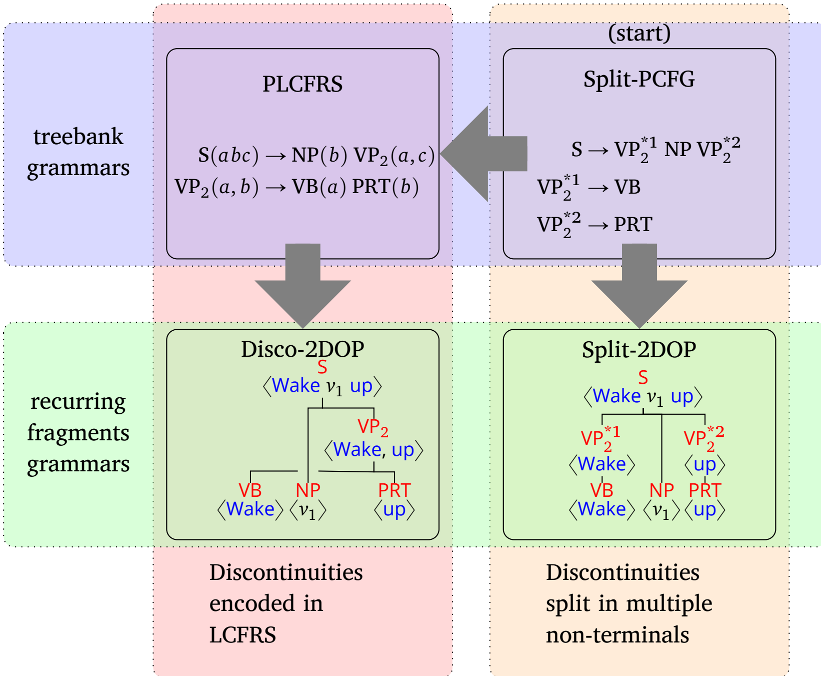


Figure 3: The systems explored in this work

aspects as disambiguation using occurrence frequencies of grammatical constructions. Thus, the choice of a formalism to describe the system's competence grammar depends on one's decisions on how syntax should be formalized. Regular and context-free grammars have been argued to be too limited (Chomsky 1956; Shieber 1985), while richer alternatives – context-sensitive and beyond – are considered too powerful to allow for an efficient computational implementation; this applies to Transformational Grammar (Peters and Ritchie 1973), Lexical-Functional Grammar, and Head-Driven Phrase Structure Grammar (Trautwein 1995). We may therefore wish to strike a balance and find a grammar formalism that is just powerful enough to describe the syntax of natural language. Joshi (1985) proposes Mildly Context-Sensitive grammars, which are beyond context-free, but avoid the computational complexity that comes with the full class of context-sensitive grammars. The first formalism developed in this framework was Tree-Adjoining Grammar (TAG; Joshi 1985). There has been work on automatic extraction of tree-adjoining grammars from corpora (Chiang 2000; Xia *et al.* 2001; Kaeshammer and Demberg 2012), and formal extensions such as multi-component TAG (Weir 1988; Schuler *et al.* 2000; Kallmeyer 2009). Linear Context-Free Rewriting Systems (LCFRSs), as employed in the work reported below, are instances of Mildly Context-Sensitive grammar. LCFRS appears to be a *lingua franca* among mildly context-sensitive formalisms, since several formalisms have been shown to be equivalent to it (Vijay-Shanker and Weir 1994).

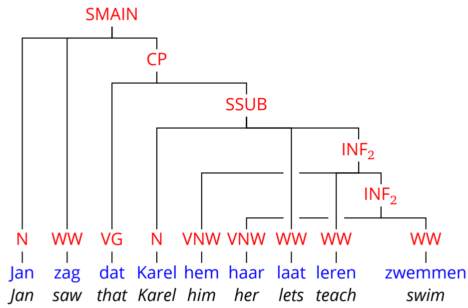
Irrespective of whether one accepts the competence-performance dichotomy, a practical natural language system needs to deal with phenomena that depend on world knowledge reflected in language use (e.g., the fact that in *"eat pizza with a fork"*, *with a fork* is prototypically related to *eat* rather than to *pizza*). This has led to a statistical turn in computational linguistics, in which models are directly induced from treebanks (Scha 1990; Charniak 1996; Bod *et al.* 2003; Geman and Johnson 2004). If the end goal is to make an adequate model of language *performance*, there is actually no need to have a competence grammar which is 'just right.' Instead, we might reduce some of the formal complexity by encoding it in statistical patterns. Concretely, we can opt for a grammar formalism that deliberately overgenerates, and count on grammatical analyses having a higher probability of being selected during disambiguation. This operationalizes the idea of there being a spectrum between ungrammaticality, markedness, and felicity. In Section 4.1 we introduce an approximation of LCFRS that makes it possible to produce discontinuous constituents in cubic time using a context-free grammar, by encoding information in non-terminal labels. A probabilistic variant of the resulting grammar makes stronger independence assumptions than the equivalent LCFRS, but as a component in a larger statistical system this does not have to pose a problem.

In the debate about the context-freeness of language, cross-serial dependencies have played an important role (Huybregts 1976; Bresnan *et al.* 1982; Shieber 1985). Consider the following example in Dutch:

(1)	Jan	zag	dat	Karel	hem	haar	laat	leren zwemmen.
	Jan	saw	that	Karel	him	her	lets	teach swim.

'Jan saw that Karel lets him teach her to swim.'

Figure 4:
Cross-serial dependencies in Dutch
expressed with discontinuous constituents



Ojeda (1988) gives an account using discontinuous constituents; cf. Figure 4. In Section 4.1 we show how such analyses may be produced by an overgenerating context-free grammar.

This is an instance of the more general idea of approximating rich formal models in formally weaker but statistically richer models, i.e., descriptive aspects of language that can be handled as a performance rather than a competence problem. Another instance of this is constituted by the various restricted versions of TAG, whose string languages form a proper subset of those of LCFRS. Restricted variants of TAG that generate context-free string languages are Tree-Insertion Grammar (Schabes and Waters 1995; Hoogweg 2003; Yamangil and Shieber 2012), and off-spine TAG (Swanson *et al.* 2013); TSG is an even more restricted variant of TAG in which the adjunction operation is removed altogether. These results suggest that there is a trade-off to be made in the choice of formalism. While on the one hand Mild Context-Sensitivity already aims to limit formal complexity to precisely what is needed for adequate linguistic description, a practical, statistical implementation presents further opportunities for constraining complexity.

Another performance aspect of language relevant for computational linguistics is pruning. While normally considered an implementation aspect made necessary by practical hardware limitations, finding linguistically and psychologically plausible shortcuts in language processing forms an interesting research question. Schuler *et al.* (2010) present a parser with human-like memory constraints based on a finite-state model. Although Roark *et al.* (2012) are not concerned with cognitive plausibility, they also work with finite-state methods and show that CFG parsing can be done in quadratic or even linear time with finite-state pruning methods.

As a specific example of a cognitive limitation relevant to parsing algorithms, consider center embedding. Karlsson (2007) reports from a corpus study that center embeddings only occur up to depth 3 in written language, and up to depth 2 in spoken language. If a statistical parser would take such cognitive limitations into account, many implausible analyses could be ruled out from the outset. More generally, it is worthwhile to strive for an explicit performance model that incorporates such cognitive and computational limitations as first class citizens.

In this work we do not go all the way to a finite-state model, but we do show that the non-local relations expressed in discontinuous constituents can be expressed in a context-free grammar model. We start with a mildly context-sensitive grammar formalism to parse discontinuous constituents, augmented with tree substitution. We then show that an approximation with context-free grammar is possible and effective. We find that the reduced independence assumptions and larger contexts taken into account as a result of tree substitution make it possible to capture non-local relations without going beyond context-free. Tree substitution thus increases the capabilities of the performance side without increasing the complexity of the competence side. A performance phenomenon that is modeled by this is that non-local relations are only faithfully produced as far as observed in the data.

3 GRAMMAR FORMALISMS

In this section we describe two formalisms related to discontinuous constituents; (string rewriting) Linear Context-Free Rewriting Systems and Discontinuous Tree-Substitution Grammar.

(String rewriting) Linear Context-Free Rewriting Systems (LCFRS; Vijay-Shanker *et al.* 1987) can produce such structures. An LCFRS generalizes CFG by allowing non-terminals to rewrite tuples of strings instead of just single, contiguous strings. This property makes LCFRS suitable for directly parsing discontinuous constituents (Kallmeyer and Maier 2010, 2013), as well as non-projective dependencies (Kuhlmann and Satta 2009; Kuhlmann 2013).

A tree-substitution grammar (TSG) provides a generalization of context-free grammar (CFG) that operates with larger chunks than just single grammar productions. A probabilistic TSG can be seen as a PCFG in which several productions may be applied at once, capturing structural relations between those productions. Tree-substitution grammars have numerous applications. They can be used for statistical parsing, such as with Data-Oriented Parsing (DOP; Scha 1990; Bod 1992; Bod *et al.* 2003; Bansal and Klein 2010; Sangati and Zuidema 2011) and Bayesian TSGs (O'Donnell *et al.* 2009; Post and Gildea 2009; Cohn *et al.* 2009, 2010; Shindo *et al.* 2012). Other applications include grammaticality judgements (Post 2011), multi-word expression identification (Green *et al.* 2011), stylometry (Bergsma *et al.* 2012; van Cranenburgh 2012b), and native language detection (Swanson and Charniak 2012).

Before defining these formalisms, we first define the tree structures they operate on. The notion of a "discontinuous tree" stems from a long linguistic tradition (Pike 1943, Sections 4.12–14; Wells 1947, Sections 55–62; McCawley 1982). It generalizes the usual notion of a phrase-structure tree in that it allows a non-terminal node to dominate a lexical span that consists of non-contiguous chunks. In our interpretation of this idea, it results in three formal differences:

1. A non-terminal with non-contiguous daughters does not have a non-arbitrary place in the left-to-right order with respect to its sibling nodes. Therefore, it is not obvious anymore that the left-to-right order of the terminals is to be described in terms of their occurrence in a tree with totally ordered branches. Instead, we employ trees with *unordered* branches, while every node is augmented with an explicit representation of its (ordered) yield.
2. An "ordinary" (totally ordered) tree has a contiguous string of leaf nodes as its yield. When we allow discontinuities, this property still applies to the (totally lexicalized) complete trees of complete sentences. But for tree fragments, it fails; their yields may contain gaps. In the general case, the yield of a discontinuous tree is thus a tuple of strings.
3. Extracting a fragment from a tree now consists of two steps:
    (a) Extracting a connected subset of nodes, and

(b) Updating the yield tuples of the nodes. In the yield tuple of every non-terminal leaf node, every element (a contiguous chunk of words) is replaced by a *terminal variable*. This replacement is percolated up the tree, to the yield tuples of all nodes. Different occurrences of the same word carry a unique index, to allow for the percolation to proceed correctly.

We now proceed to give a more formal definition of our notion of a discontinuous tree.

DEFINITION 1. A *discontinuous syntactic tree* is a rooted, unordered tree. Each node consists of a label and a yield. A yield is a tuple of strings composed of lexical items; the tuple of strings denotes a subsequence of the yield at the root of the tree. We write ⟨a b⟩ to denote a yield consisting of the contiguous sequence of lexical items 'a' and 'b', while ⟨a b, c⟩ denotes a yield containing 'a b' followed by 'c' with an intervening gap. Given a node X,

- the yield of X is composed of the terminals in the yields of the children of X;
- conversely, the yield of each child of X is a subsequence of the yield of X;
- the yields of siblings do not overlap.

Figure 5 shows a tree according to this definition in which discontinuities are visualized with crossing branches as before. The same tree is rendered in Figure 6, without crossing branches, to highlight the fact that the information about discontinuities is encoded in the yields of the tree nodes.

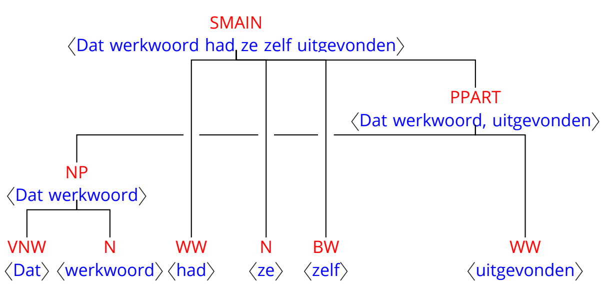


Figure 5:
A discontinuous tree with yield tuples

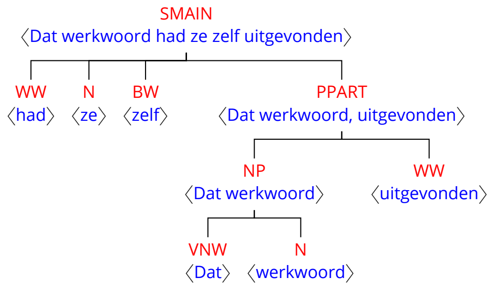


Figure 6: An equivalent representation of the tree in Figure 5, without crossing branches

Definition 2. An *incomplete tree* is a discontinuous tree in which the yields may contain variables v_n with $n \in \mathbb{N}$ in addition to lexical items. Variables stand in for any contiguous string of lexical items. An incomplete tree contains 2 or more nodes, or a single node with only lexical items in its yield. A node without children whose yield consists solely of variables is called a *substitution site*.

An incomplete tree may be derived from an extracted tree fragment. The tree fragment may contain variables for substrings which needed to be distinguished in other parts of the tree, but only occur contiguously in the fragment. We reduce these strings of contiguous variables to single variables; i.e., we abstract fragments from their original context by reducing strings of variables that appear contiguously across the fragment into single variables (e.g. Figure 7).

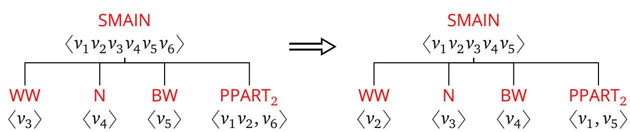


Figure 7: Reducing variables in a fragment extracted from the tree in Figure 5

The *fan-out* of a non-terminal node equals the number of terminals in its yield that are not directly preceded by another terminal in the same yield; i.e., the number of contiguous substrings (components) of which the yield consists.[1] From here on we denote the fan-out of a discontinuous non-terminal with a subscript that is part of its label.

[1] Note that a distinction is often made between the *fan-out* of non-terminals in grammar productions, and the *block degree* of nodes of a syntactic tree (Maier and Lichte 2011; Kuhlmann 2013). Due to the fact that the productions of a TSG are trees, these notions coincide for our purposes.

3.1 Linear Context-Free Rewriting Systems

String-rewriting LCFRS can be seen as the discontinuous counterpart of CFG, and its probabilistic variant can be used to articulate a discontinuous treebank grammar. LCFRS productions differ from CFG productions in that they generate for a given non-terminal one or more strings at a time in potentially non-adjacent positions in the sentence. The number of these positions, the measure of discontinuity in a constituent, is called the fan-out. A CFG is an LCFRS with a maximum fan-out of 1. Together with the number of non-terminals on the right-hand side, the fan-out defines a hierarchy of grammars with increasing complexity, of which CFG is the simplest case. In this paper we use the simple RCG notation (Boullier 1998) for LCFRS. We focus on string-rewriting LCFRS and use the tree produced as a side-effect of a string's derivation as its syntactic analysis. It is possible to define an LCFRS that rewrites trees or graphs; however, the formalisms used in this paper are all expressible as string-rewriting LCFRSs.

DEFINITION 3. A string-rewriting LCFRS is a tuple $G = \langle N, T, V, P, S \rangle$. N and T are disjoint finite sets of non-terminals and terminals, respectively. A function $\varphi : N \to \{1, 2, \ldots, \}$ specifies the unique fan-out for every non-terminal symbol. V is a finite set of variables; we refer to the variables as x^i_j with $i, j \in \mathbb{N}$. S is the distinguished start symbol with $S \in N$ and $\varphi(S) = 1$. P is a finite set of productions, of the form:

$$A(\alpha_1, \ldots, \alpha_{\varphi(A)}) \to B_1(x^1_1, \ldots, x^1_{\varphi(B_1)}) \ldots B_r(x^r_1, \ldots, x^r_{\varphi(B_r)})$$

for $r \geqslant 0$, where A, B_1, ..., $B_r \in N$, each $x^i_j \in V$ for $1 \leqslant i \leqslant r$, $1 \leqslant j \leqslant \varphi(B_i)$, and $\alpha_j \in (T \cup V)^+$ for $1 \leqslant j \leqslant \varphi(A)$. Observe that a component α_j is a concatenation of one or more terminals and variables.

The rank r refers to the number of non-terminals on the right-hand side of a production, while the fan-out φ of a non-terminal refers to the number of components it covers. A rank of zero implies a lexical production; in that case the right-hand side (RHS) is notated as ε implying no new non-terminals are produced (not to be confused with generating the empty string), and the left-hand side (LHS) argument is composed only of terminals.

Productions must be *linear* and *non-erasing*: if a variable occurs in a production, it occurs exactly once on the LHS, and exactly once on the RHS. A production is *monotone*[2] if for any two variables x_1 and x_2 occurring in a non-terminal on the RHS, x_1 precedes x_2 on the LHS iff x_1 precedes x_2 on the RHS. Due to our method of grammar extraction from treebanks, (cf. Section 3.1.1 below) all productions in this work are monotone and, except in some examples, at most binary ($r \leqslant 2$); lexical productions ($r = 0$) have fan-out 1 and introduce only a single terminal.

A production is *instantiated* when its variables are bound to spans such that for each component α_j of the LHS, the concatenation of the strings that its terminals and bound variables point to forms a contiguous, non-overlapping span in the input. In the remainder we will notate discontinuous non-terminals with a subscript indicating their fan-out.

When a sentence is parsed by an LCFRS, its derivation tree (Boullier 1998, Section 3.3; Kallmeyer 2010, pp. 115–117) is a discontinuous tree. Conversely, given a set of discontinuous trees, a set of productions can be extracted that generate those trees.

In a probabilistic LCFRS (PLCFRS), each production is associated with a probability and the probability of derivation is the product of the probabilities of its productions. Analogously to a PCFG, a PLCFRS may be induced from a treebank by using relative frequencies as probabilities (Maier and Søgaard 2008).

DEFINITION 4. The *language* of an LCFRS G is defined as follows (Kallmeyer and Maier 2013, pp. 92–93):

1. For every $A \in N$, we define the yield of A, $\text{yield}_G(A)$, as follows:
   (a) For every production $A(t) \rightarrow \varepsilon$ with $t \in T$, $\langle t \rangle \in \text{yield}_G(A)$
   (b) For every production

$$A(\alpha_1, \ldots, \alpha_{\varphi(A)}) \rightarrow B_1(x_1^1, \ldots, x_{\varphi(B_1)}^1) \ldots B_r(x_1^r, \ldots, x_{\varphi(B_r)}^r)$$

   and all tuples $\tau_1 \in \text{yield}_G(B_1), \ldots, \tau_r \in \text{yield}_G(B_r)$:

$$\langle f(\alpha_1), \ldots, f(\alpha_{\varphi(A)}) \rangle \in \text{yield}_G(A)$$

   where f is defined as follows:
   i. $f(t) = t$ for all $t \in T$,

[2] This property is called *ordered* in the RCG literature.

ii. $f(x_j^i) = \tau_i[j]$ for all $1 \leqslant i \leqslant r, 1 \leqslant j \leqslant \varphi(B_i)$, and

iii. $f(ab) = f(a)f(b)$ for all $a, b \in (T \cup V)^+$.

f is the *composition function* of the production.

(c) Nothing else is in $\text{yield}_G(A)$.

2. The language of G is then $L(G) = \text{yield}_G(S)$.

3.1.1 Extracting LCFRS productions from trees

LCFRS productions may be induced from a discontinuous tree, using a procedure described in Maier and Søgaard (2008). We extend this procedure to handle substitution sites, i.e., non-terminals with only variable terminals in their yield, but no lexical items; such nodes occur in tree fragments extracted from a treebank. The procedure is as follows:

Given a discontinuous tree, we extract a grammar production for each non-leaf non-terminal node. The label of the node forms the LHS non-terminal, and the labels of the nodes immediately dominated by it form the RHS non-terminals. The arguments of each RHS non-terminal are based on their yield tuples. Adjacent variables in the yield of the RHS non-terminals are collapsed into single variables and replaced on both LHS and RHS. Consider the tree fragment in Figure 7, which gives the following LCFRS production:

$$\text{SMAIN}(abcde) \rightarrow \text{PPART}(a, e)\ \text{WW}(b)\ \text{N}(c)\ \text{BW}(d)$$

Pre-terminals yield a production with their terminal as a direct argument to the pre-terminal, and an empty RHS. Substitution sites in a tree only appear on the RHS of extracted productions, since it is not known what they will expand to. See Figure 8 for examples of LCFRS productions extracted from a discontinuous tree.

3.2 *Discontinuous Tree-Substitution Grammar*

We now employ string-rewriting LCFRS, introduced in the previous section, to replace the CFG foundation of TSGs. Note that the resulting formalism directly rewrites elementary trees with discontinuous constituents, making it an instantiation of the more general notion of a tree-rewriting LCFRS. Tree-rewriting LCFRSs are more general because they allow other rewriting operations besides substitution. However, since we limit the operations in the formalism

Figure 8: The LCFRS $G = \langle N, T, V, P, S \rangle$ extracted from the tree in Figure 5

$$
\begin{aligned}
N &= \{\text{SMAIN, PPART, NP, VNW, N, WW, BW}\} \\
T &= \{\text{Dat, had, uitgevonden, werkwoord, ze, zelf}\} \\
V &= \{a, b, c, d, e\} \\
\varphi &= \{\text{SMAIN} : 1, \text{PPART} : 2, \text{NP} : 1, \\
&\quad \text{VNW} : 1, \text{N} : 1, \text{WW} : 1, \text{BW} : 1\} \\
S &= \text{SMAIN} \\
P &= \{\text{SMAIN}(abcde) \rightarrow \text{WW}(b)\ \text{N}(c)\ \text{BW}(d)\ \text{PPART}(a, e), \\
&\quad \text{PPART}(a, b) \rightarrow \text{NP}(a)\ \text{WW}(b), \\
&\quad \text{NP}(ab) \rightarrow \text{VNW}(a)\ \text{N}(b), \\
&\quad \text{VNW}(\text{Dat}) \rightarrow\ \varepsilon,\ \ \text{N}(\text{werkwoord}) \rightarrow\ \varepsilon, \\
&\quad \text{WW}(\text{had}) \rightarrow\ \varepsilon,\ \ \text{N}(\text{ze}) \rightarrow\ \varepsilon,\ \ \text{BW}(\text{zelf}) \rightarrow\ \varepsilon, \\
&\quad \text{WW}(\text{uitgevonden}) \rightarrow\ \varepsilon\}
\end{aligned}
$$

to substitution, it remains possible to specify a direct mapping to a string-rewriting grammar, as we shall see in the next section. As noted before, a TSG can be seen as a TAG without the adjunction operation. A discontinuous TSG may be related to a special case of set-local multi-component TAG (Weir 1988; Kallmeyer 2009). A multi-component TAG is able to specify constraints that require particular elementary trees to apply together; this mechanism can be used to generate the non-local elements of discontinuous constituents.

The following definitions are based on the definition for continuous TSG in Sima'an (1997).

DEFINITION 5. A *probabilistic, discontinuous TSG* (PDTSG) is a tuple $\langle N, T, V, S, \mathscr{C}, P \rangle$, where N and T are disjoint finite sets that denote the set of non-terminal and terminal symbols, respectively; V is a finite set of variables; S denotes the start non-terminal; and $\mathscr{C}$ is a finite set of elementary trees. For all trees in $\mathscr{C}$ it holds that for each non-terminal, there is a unique fan-out; this induces a function $\varphi \subset N \times \{1, 2, \ldots\}$ with $\varphi(A)$ being the unique fan-out of $A \in N$. For convenience, we abbreviate $\varphi(\text{root}(t))$ for a tree t as $\varphi(t)$. The function P assigns a value $0 < P(t) \leqslant 1$ (probability) to each elementary tree t such that for every non-terminal $A \in N$, the probabilities of all elementary trees whose root node is labelled A sum to 1.

The tuple $\langle N, T, V, S, \mathscr{C} \rangle$ of a given PDTSG $\langle N, T, V, S, \mathscr{C}, P \rangle$ is called the DTSG underlying the PDTSG.

DEFINITION 6. *Substitution:* The substitution $A \circ B$ is defined iff the label of the left-most substitution site of A equals the label of the root node of B. The left-most substitution site of an incomplete tree A is the leaf node containing the first occurrence of a variable in the yield of the root of A. When defined, the result of $A \circ B$ equals a copy of the tree A with B substituted for the left-most substitution site of A. In the yield argument of A, each variable terminal is replaced with the corresponding component of one or more contiguous terminals from B. For example, given yield$(A) = \langle l_1 v_2, l_4 \rangle$ and yield$(B) = \langle l_2 l_3 \rangle$ where l_n is a lexical terminal and v_n a variable, yield$(A \circ B) = \langle l_1 l_2 l_3, l_4 \rangle$.

DEFINITION 7. A *left-most derivation* (derivation henceforth) d is a sequence of zero or more substitutions $T = (\ldots(f_1 \circ f_2) \circ \ldots) \circ f_m$, where $f_1, \ldots, f_m \in \mathscr{C}$, root$(T)$ = root$(f_1) = S$, $\varphi(T) = 1$ and T contains no substitution sites. The probability $P(d)$ is defined as:

$$P(f_1) \cdot \ldots \cdot P(f_m) = \prod_{i=1}^{m} P(f_i)$$

Refer to Figure 9 for an example.

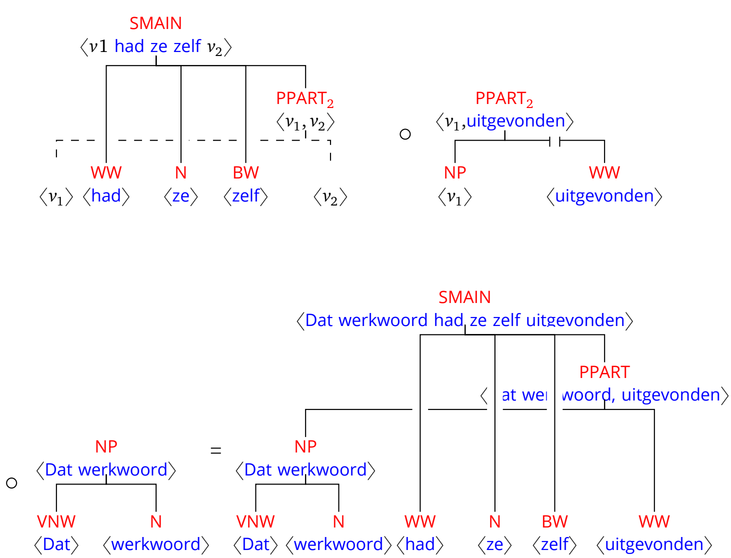


Figure 9: A discontinuous tree-substitution derivation of the tree in Figure 1. Note that in the first fragment, which has a discontinuous substitution site, the destination for the discontinuous spans is marked in advance, shown with variables (v_n) as placeholders.

DEFINITION 8. A *parse* is any tree which is the result of a derivation. A parse can have various derivations. Given the set $D(T)$ of derivations yielding parse T, the probability of T is defined as $\sum_{d \in D(T)} P(d)$.

4 GRAMMAR TRANSFORMATIONS

CFG, LCFRS, and DTSG can be seen as natural extensions of each other. This makes it possible to define transformations that help to make parsing more efficient. Specifically, we define simplified versions of these grammars that can be parsed efficiently, while their productions or labels map back to the original grammar.

4.1 *A CFG approximation of discontinuous LCFRS parsing*

Barthélemy *et al.* (2001) introduced a technique to guide the parsing of a range concatenation grammar (RCG) by a grammar with a lower parsing complexity. Van Cranenburgh (2012a) applies this idea to probabilistic LCFRS parsing and extends the method to prune unlikely constituents in addition to filtering impossible constituents.

The approximation can be formulated as a tree transformation instead of a grammar transformation. The tree transformation by Boyd (2007) encodes discontinuities in the labels of tree nodes.[3] The resulting trees can be used to induce a PCFG that can be viewed as an approximation to the corresponding PLCFRS grammar of the original, discontinuous treebank. We will call this a Split-PCFG.

DEFINITION 9. A Split-PCFG is a PCFG induced from a treebank transformed by the method of Boyd (2007); that is, discontinuous constituents have been split into several non-terminals, such that each new non-terminal covers a single contiguous component of the yield of the discontinuous constituent. Given a discontinuous non-terminal

[3] Hsu (2010) compares three methods for resolving discontinuity in trees: (*a*) node splitting, as applied here; (*b*) node adding, a simpler version of node splitting that does not introduce new non-terminal labels; and (*c*) node raising, the more commonly applied method of resolving discontinuity. While the latter two methods yield better performance, we use the node splitting approach because it provides a more direct mapping to discontinuous constituents, which, as we shall later see, makes it a useful source of information for pruning purposes.

X_n in the original treebank, the new non-terminals will be labelled X_n^{*m}, with m the index of the component, s.t. $1 \leqslant m \leqslant n$.

For example:

$$\begin{aligned} \text{LCFRS productions: } \mathrm{S}(abc) &\to \mathrm{NP}(b)\ \mathrm{VP}_2(a,c) \\ \mathrm{VP}_2(a,b) &\to \mathrm{VB}(a)\ \mathrm{PRT}(b) \\ \text{CFG approximation: } \mathrm{S} &\to \mathrm{VP}_2^{*1}\ \mathrm{NP}\ \mathrm{VP}_2^{*2} \\ \mathrm{VP}_2^{*1} &\to \mathrm{VB} \\ \mathrm{VP}_2^{*2} &\to \mathrm{PRT} \end{aligned}$$

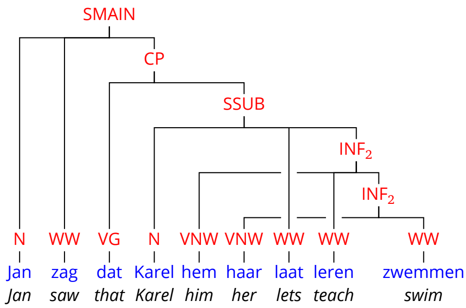
In a post-processing step, PCFG derivations are converted to discontinuous trees by merging siblings marked with '*'. This approximation overgenerates compared to the respective LCFRS, i.e., it licenses a superset of the derivations of the respective LCFRS. For example, a component VP_2^{*1} may be generated without generating its counterpart VP_2^{*2}; such derivations can be filtered in post-processing. Furthermore, two components VP_2^{*1} and VP_2^{*2} may be generated which were extracted from different discontinuous constituents, such that their combination could not be generated by the LCFRS.[4] Another problem would occur when productions contain discontinuous constituents with the same label; the following two productions map to the same productions in the CFG approximation:

$$\begin{aligned} \mathrm{VP}(adceb) &\to \mathrm{VP}_2(a,b)\ \mathrm{CNJ}(c)\ \mathrm{VP}_2(d,e) \\ \mathrm{VP}(adcbe) &\to \mathrm{VP}_2(a,b)\ \mathrm{CNJ}(c)\ \mathrm{VP}_2(d,e) \end{aligned}$$

However, such productions do not occur in any of the treebanks used in this work. The increased independence assumptions due to rewriting discontinuous components separately are more problematic, especially with nested discontinuous constituents. They necessitate the use of non-local statistical information to select the most likely structures, for instance by turning to tree-substitution grammar (cf. Section 2 above). (Note that the issue is not as problematic when the approximation is only used as a source of pruning information).

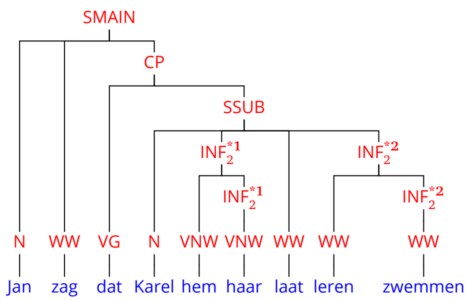
As a specific example of the transformation, consider the case of cross-serial dependencies. Figure 10 shows the parse tree for the

[4] A reviewer points out that if discontinuous rewriting is seen as synchronous rewriting (synchronous CFGs are equivalent to LCFRSs with fan-out 2), the split transformation is analogous to taking out the synchronicity.



discontinuous constituents:

$$\begin{aligned}
\mathrm{SMAIN}(abc) &\rightarrow \mathrm{N}(a)\ \mathrm{WW}(b)\ \mathrm{CP}(c)\\
\mathrm{CP}(ab) &\rightarrow \mathrm{VG}(a)\ \mathrm{SSUB}(b)\\
\mathrm{SSUB}(abcd) &\rightarrow \mathrm{N}(a)\ \mathrm{INF}_2(b,d)\ \mathrm{WW}(c)\\
\mathrm{INF}_2(ab,cd) &\rightarrow \mathrm{VNW}(a)\ \mathrm{INF}_2(b,d)\ \mathrm{WW}(c)\\
\mathrm{INF}_2(a,b) &\rightarrow \mathrm{VNW}(a)\ \mathrm{WW}(b)
\end{aligned}$$



PCFG approximation:

$$\begin{aligned}
\mathrm{SMAIN} &\rightarrow \mathrm{N\ WW\ CP}\\
\mathrm{CP} &\rightarrow \mathrm{VG\ SSUB}\\
\mathrm{SSUB} &\rightarrow \mathrm{N\ INF}_2^{*1}\ \mathrm{WW\ INF}_2^{*2}\\
\mathrm{INF}_2^{*1} &\rightarrow \mathrm{VNW\ INF}_2^{*1}\\
\mathrm{INF}_2^{*2} &\rightarrow \mathrm{WW\ INF}_2^{*2}\\
\mathrm{INF}_2^{*1} &\rightarrow \mathrm{VNW}\\
\mathrm{INF}_2^{*2} &\rightarrow \mathrm{WW}
\end{aligned}$$

Figure 10: Cross-serial dependencies in Dutch expressed with discontinuous constituents (top); and the same parse tree, after discontinuities have been encoded in node labels (bottom)

example sentence from the previous section, along with the grammar productions for it, before and after applying the CFG approximation of LCFRS. Note that in the approximation, the second level of INF nodes may be rewritten separately, and a context-free grammar cannot place the non-local constraint that each transitive verb should be paired with a direct object. On the other hand, through the use of tree substitution, an elementary tree may capture the whole construction of two verbs cross-serially depending on two objects, and the model needs only to prefer an analysis with this elementary tree. Once an elementary tree contains the whole construction, it no longer matters whether its internal nodes contain discontinuous constituents or indexed node labels, and the complexity of discontinuous rewriting is weakened to a statistical regularity.

A phenomenon which cannot be captured in this representation, not even with the help of tree-substitution, is recursive synchronous rewriting (Kallmeyer *et al.* 2009). Although this phenomenon is rare, it does occur in treebanks.

4.2 TSG compression

Using grammar transformations, it is possible to parse with a TSG without having to represent elementary trees in the chart explicitly, but instead work with a parser for the base grammar underlying the TSG (typically a CFG, in our case an LCFRS).

In this section we present such a transformation for an arbitrary discontinuous TSG to a string-rewriting LCFRS. We first look at well-established strategies for reducing a continuous TSG to a CFG, and then show that these carry over to the discontinuous case. Previous work was based on probabilistic TSG without discontinuity; this special case of PDTSG is referred to as PTSG.

4.2.1 Compressing PTSG to PCFG

Goodman (2003) gives a reduction to a PCFG for the special case of a PTSG based on all fragments from a given treebank and their frequencies. This reduction is stochastically equivalent to an all-fragments PTSG after the summation of probabilities from equivalent derivations; however, it does not admit parsing with TSGs consisting of arbitrary sets of elementary trees or assuming arbitrary probability models. Perhaps counter-intuitively, restrictions on the set of fragments increase the size of Goodman's reduction (e.g., depth restriction, Goodman 2003, p. 134). While Goodman (2003) gives instantiations of his reduction with various probability models, the limitation is that probability assignments of fragments have to be expressible as a composition of the weights of the productions in each fragment. Since each production in the reduction participates in numerous implicit fragments, it is not possible to adjust the probability of an individual fragment without affecting related fragments. We leave Goodman's reduction aside for now, because we would prefer a more general method.

A naive way to convert any TSG is to decorate each internal node of its elementary trees with a globally unique number, which can be removed from derivations in a post-processing step. Each elementary tree then contributes one or more grammar productions, and because of the unique labels, elementary trees will always be derived as a whole. However, this conversion results in a large number of non-terminals, which are essentially 'inert': they never participate in substitution but deterministically rewrite to the rest of their elementary tree.

A more compact transformation is used in Sangati and Zuidema (2011), which can be applied to arbitrary PTSGs, but adds a minimal number of new non-terminal nodes. Internal nodes are removed from elementary trees, yielding a flattened tree of depth 1. Each flattened tree is then converted to a grammar production. Each production and original fragment is stored in a backtransform table. This table makes it possible to restore the original fragments of a derivation built from flattened productions. Whenever two fragments would map to the same flattened production, a unary node with a unique identifier is added to disambiguate them. The weight associated with an elementary tree carries over to the first production it produces; the rest of the productions are assigned a weight of 1.

4.2.2 Compressing PDTSG to PLCFRS

The transformation defined by Sangati and Zuidema (2011) assumes that a sequence of productions can be read off from a syntactic tree, such as a standard phrase-structure tree that can be converted into a sequence of context-free grammar productions. Using the method for inducing LCFRS productions from syntactic trees given in Section 4.2.1, we can apply the same TSG transformation to discontinuous trees as well.

Due to the design of the parser we will use, it is desirable to have grammar productions in binarized form, and to separate phrasal and lexical productions. We therefore binarize the flattened trees with a left-factored binarization that adds unique identifiers to every intermediate node introduced by the binarization. In order to separate phrasal and lexical productions, a new POS tag is introduced for each terminal, which selects for that specific terminal. A sequence of productions is then read off from the transformed tree. The unique identifier in the first production is used to look up the original elementary tree in the backtransform table.[5]

Figure 11 illustrates the transformation of a discontinuous TSG. The middle column shows the productions after transforming each ele-

[5] Note that only this first production requires a globally unique identifier; to reduce the grammar constant, the other identifiers can be merged for equivalent productions.

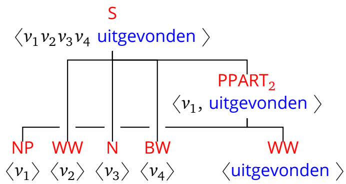
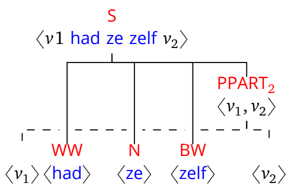
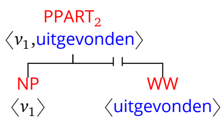
Elementary tree	Productions	Weight
	$\mathrm{S}(ab) \rightarrow \mathrm{S}^1(a)\ \mathrm{WW}(b)$	f/f'
	$\mathrm{S}^1(ab) \rightarrow \mathrm{S}^2(a)\ \mathrm{BW}(b)$	1
	$\mathrm{S}^2(ab) \rightarrow \mathrm{S}^3(a)\ \mathrm{N}(b)$	1
	$\mathrm{S}^3(ab) \rightarrow \mathrm{NP}(a)\ \mathrm{WW}^4(b)$	1
	$\mathrm{WW}^4(\mathrm{uitgevonden}) \rightarrow \varepsilon$	1
	$\mathrm{S}(abc) \rightarrow \mathrm{S}^5_2(a,c)\ \mathrm{BW}^6(b)$	f/f'
	$\mathrm{S}^5_2(ab,c) \rightarrow \mathrm{S}^7_2(a,c)\ \mathrm{N}(b)$	1
	$\mathrm{S}^7_2(ab,c) \rightarrow \mathrm{PPART}_2(a,c)\ \mathrm{WW}^8(b)$	1
	$\mathrm{WW}^8(\mathrm{had}) \rightarrow \varepsilon$	1
	$\mathrm{N}^7(\mathrm{ze}) \rightarrow \varepsilon$	1
	$\mathrm{BW}^6(\mathrm{zelf}) \rightarrow \varepsilon$	1
	$\mathrm{PPART}_2(a,b) \rightarrow \mathrm{NP}(a)\ \mathrm{WW}^9(b)$	f/f'
	$\mathrm{WW}^9(\mathrm{uitgevonden}) \rightarrow \varepsilon$	1

Figure 11: Transforming a discontinuous tree-substitution grammar into an LCFRS and backtransform table. The elementary trees are extracted from the tree in Figure 1 with labels abbreviated. The first production of each fragment is used as an index to the backtransform table so that the original fragments in derivations can be reconstructed.

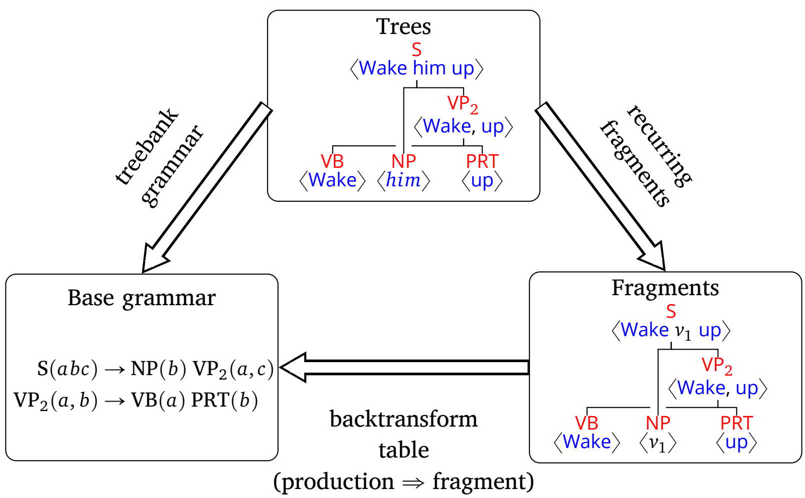


Figure 12: Diagram of the methods of grammar induction.

mentary tree. The rightmost column shows how relative frequencies can be used as weights, where f is the frequency of the elementary tree in the treebank, and f' is the frequency mass of elementary trees with the same root label. Note that the productions for the first elementary tree contain no discontinuity, because the discontinuous internal node is eliminated. Conversely, the transformation may also introduce more discontinuity, due to the binarization (but cf. Section 8.1 below).

Figure 12 presents an overview of the methods of grammar induction presented thus far, as well as the approach for finding recurring fragments that will be introduced in the next section.

5 INDUCING A TSG FROM A TREEBANK

In Data-Oriented Parsing the grammar is implicit in the treebank itself, and in principle all possible fragments from its trees can be used to derive new sentences. Grammar induction is therefore conceptually simple (even though the grammar may be very large), as there is no training or learning involved. This maximizes re-use of previous experience.

The use of all possible fragments allows for multiple derivations of the same tree; this spurious ambiguity is seen as a virtue in DOP, because it combines the specificity of larger fragments and the smoothing of smaller fragments. This is in contrast to parsimonious approaches which decompose each tree in the training corpus into a sequence of fragments representing a single derivation.

5.1 *Extracting recurring fragments*

Representing all possible fragments of a treebank is not feasible, since the number of fragments is exponential in terms of the number of nodes. A practical solution is to define a subset. A method called Double-DOP (2DOP; Sangati and Zuidema 2011) implements this without compromising on the principle of data-orientation. It restricts the fragment set to recurring fragments, i.e., fragments that occur in at least two different contexts. These are found by considering every pair of trees and extracting the largest tree fragments they have in common. It is feasible to do this exhaustively for the whole treebank. This is in contrast to the sampling of fragments in earlier DOP models (Bod 2001) and Bayesian TSGs. Since the space of fragments is enormous (that is, exponential in terms of sentence length), it stands to reason that a sampling approach will not discover all relevant fragments in a reasonable time frame.

Sangati *et al.* (2010) presents a tree-kernel method for extracting maximal recurring fragments that operates in quadratic time in terms of the number of nodes in the treebank. A faster version of this method was presented in van Cranenburgh (2014), which uses a linear average time tree kernel, and introduces the ability to handle discontinuous trees. We obtain a further increase in speed by implementing an inverted index with a compressed bitmap (Chambi *et al.* 2015).

5.2 *Discontinuous fragments*

The aforementioned fragment extraction algorithms can be adapted to support trees with discontinuous constituents. Instead of implementing a new version with data structures for discontinuous trees following Definitions 1 and 2, we apply a representation that makes it possible to add discontinuous trees as a special case.

In the representation, leaf nodes are decorated with indices indicating their ordering. Just as in Figure 6, a discontinuous tree may be represented as a continuous tree, as long as information about the yield is encoded somehow. We do this by storing indices as leaf nodes, which denote an ordering and refer to a separate list of tokens. This makes it possible to use the same data structures as for continuous trees, as long as the child nodes are kept in a canonical order (induced from the order of the lowest index of each child).

Indices are used not only to keep track of the order of lexical nodes in a fragment, but also for that of the contribution of substitution sites. This is necessary in order to preserve the configuration of the yield in the original sentence. When leaf nodes are compared, the indices stand in for the token at the sentence position referred to. After a fragment is extracted, any indices need to be canonicalized. The indices originate from the original sentence, but need to be decoupled from this original context. This process is analogous to how LCFRS productions are read off from a tree with discontinuous constituents, in which contiguous intervals of indices are replaced by variables.

The canonicalization of fragments is achieved in three steps, as defined in the pseudocode of Algorithm 1; Figure 13 illustrates the process. In the examples, substitution sites have spans denoted with inclusive *start:end* intervals, as extracted from the original parse tree, which are reduced to variables denoting contiguous spans whose relation to the other spans is reflected by their indices.

Algorithm 1 Canonicalizing discontinuous fragments.

INPUT: A tree fragment t with indexed terminals w_i or intervals $\langle i : j, \dots \rangle$ as leaves ($0 \leqslant i < j < n$)
OUTPUT: A tree fragment with modified indices.

$k \leftarrow$ the smallest index in t
subtract k from each index in t
FOR ALL intervals I $= \langle i : j, \dots \rangle$ of the substitution sites in t
    FOR ALL $i : j \in I$
        replace $i : j$ with i
        subtract $j - i$ from all indices k s.t. $k > j$
FOR ALL indices i in t
    IF the indices $i + 1$ and $i + 2$ are not in t
        $k \leftarrow$ the smallest index in t s.t. $k > i$
        subtract $k - i$ from all indices y s.t. $y > i$

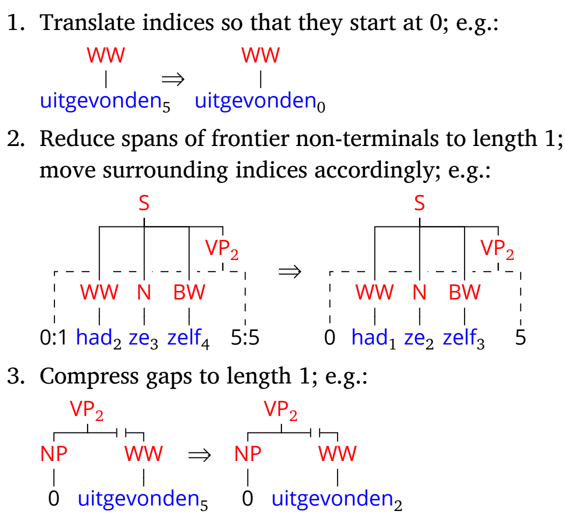


Figure 13: Canonicalization of fragments extracted from parse trees. These sample fragments have been extracted from the tree in Figure 1. The fragments are visualized here as discontinuous tree structures, but since the discontinuities are encoded in the indices of the yield, they can be represented in a standard bracketing format as used by the fragment extractor.

We will refer to the combination of Double-DOP with discontinuous constituents as Disco-2DOP. When recurring fragments are extracted from the Tiger treebank (cf. Section 8.1), we find that 10.4%

of fragment types contain a discontinuous node (root, internal, or substitution site). This can be contrasted with the observation that 30% of sentences in the Tiger treebank contain one or more discontinuous constituents, and that 20.9% of production types in the PLCFRS treebank grammar of Tiger contain a discontinuous non-terminal. On the other hand, when occurrence frequencies are taken into account, both the fragments and productions with discontinuities account for around 6.5% of the total frequency mass.

6 PARSING WITH PLCFRS AND PDTSG

After extracting fragments by means of the method of Section 5, we augment the set of fragments with all depth 1 fragments, in order to preserve complete coverage of the training set trees. Since depth 1 fragments are equivalent to single grammar productions, this ensures strong equivalence between the TSG and the respective treebank grammar.[6] We then apply the grammar transformation (cf. Section 4.2.1) to turn the fragments into productions. Productions corresponding to fragments are assigned a probability based on the relative frequency of the respective fragment; productions introduced by the transformation are given a probability of 1. For an example, please refer back to Figure 11.

We parse with the transformed grammar using the `disco-dop` parser (van Cranenburgh *et al.* 2011; van Cranenburgh 2012a). This is an agenda-based parser for PLCFRS based on the algorithm in Kallmeyer and Maier (2010, 2013), extended to produce n-best derivations (Huang and Chiang 2005) and exploit coarse-to-fine pruning (Charniak *et al.* 2006).

Parsing with LCFRS can be done with a weighted deduction system and an agenda-based parser. The deduction steps are given in Figure 14; for the pseudo-code of the parser see Algorithm 2, which is an extended version of the parser in Kallmeyer and Maier (2010, 2013) that obtains the complete parse forest as opposed to just the Viterbi derivation.

[6] Previous DOP work such as Zollmann and Sima'an (2005) adds all possible tree fragments up to depth 3. Preliminary experiments on 2DOP gave no improvement on performance, while tripling the grammar size; therefore we do not apply this in further experiments.

Figure 14: Weighted deduction system for binarized LCFRS

Lexical: $$\frac{}{p : [A, \langle\langle w_i \rangle\rangle]} \qquad p : A(w_i) \rightarrow \varepsilon \in \mathscr{G}$$

Unary: $$\frac{x : [B, \alpha]}{p \cdot x : [A, \alpha]} \qquad p : A(\alpha) \rightarrow B(\alpha)$$ is an instantiated rule from $\mathscr{G}$

Binary: $$\frac{x : [B, \beta],\ y : [C, \gamma]}{p \cdot x \cdot y : [A, \alpha]} \qquad p : A(\alpha) \rightarrow B(\beta)\ C(\gamma)$$ is an instantiated rule from $\mathscr{G}$

Goal: $$[S, \langle\langle w_1 \cdots w_n \rangle\rangle]$$

In Section 6.1 we describe the probabilistic instantiation of DTSG and the criterion to select the best parse. Section 6.2 describes how derivations from the compressed TSG are converted back into trees composed of the full elementary trees. Section 6.4 describes how coarse-to-fine pruning is employed to make parsing efficient.

Algorithm 2 A probabilistic agenda-based parser for LCFRS.

INPUT: A sentence $w_1 \cdots w_n$, a grammar $\mathscr{G}$
OUTPUT: A chart $\mathscr{C}$ with Viterbi probabilities, a parse forest $\mathscr{F}$.

```
initialize agenda 𝒜 with all possible POS tags for input
WHILE 𝒜 not empty
    ⟨I, x⟩ ← pop item with best score on agenda
    add ⟨I, x⟩ to 𝒞
    FOR ALL ⟨I′, z⟩ that can be deduced from ⟨I, x⟩ and items in 𝒞
        IF I′ ∉ 𝒜 ∪ 𝒞
            enqueue ⟨I′, z⟩ in 𝒜
        ELSE IF I′ ∈ 𝒜 ∧ z > score for I′ in 𝒜
            update weight of I′ in 𝒜 to z
        add edge for I′ to ℱ
```

6.1 *Probabilities and disambiguation*

Our probabilistic model uses the relative frequency estimate (RFE), which has shown good results with the Double-DOP model (Sangati and Zuidema 2011). The relative frequency of a fragment is the number of its occurrences, divided by the total number of occurrences of fragments with the same root node.

In DOP many derivations may produce the same parse tree, and it has been shown that approximating the most probable parse, which considers all derivations for a tree, yields better results than the most probable derivation (Bod 1995). To select a parse tree from a derivation forest, we compute tree probabilities on the basis of the 10,000 most probable DOP derivations, and select the tree with the largest probability. Although the algorithm of Huang and Chiang (2005) makes it is possible to extract the exact k-best derivations from a derivation forest, we apply pruning while building the forest.

6.2 *Reconstructing derivations*

After a derivation forest is obtained and a list of k-best derivations has been produced, the backtransform is applied to these derivations to recover their internal structure. This proceeds by doing a depth-first traversal of the derivations, and expanding each non-intermediate[7] node into a template of the original fragment. These templates are stored in a backtransform table indexed by the first binarized production of the fragment in question. The template fragment has its substitution sites marked, which are filled with values obtained by recursively expanding the children of the current constituent.

6.3 *Efficient discontinuous parsing*

We review several strategies for making discontinuous parsing efficient. As noted by Levy (2005, p. 138), the intrinsic challenge of discontinuous constituents is that a parser will generate a large number of potential discontinuous spans.

6.3.1 Outside estimates

Outside estimates (also known as context-summary estimates and figures-of-merit) are computed offline for a given grammar. During parsing they provide an estimate of the outside probability for a given constituent, i.e., the probability of a complete derivation with that constituent divided by the probability of the constituent. The estimate can be used to prioritize items in the agenda. Estimates were first introduced for discontinuous LCFRS parsing in Kallmeyer and Maier (2010, 2013). Their estimates are only applied up to sentences of 30 words. Beyond 30 words the table grows too large.

[7] An intermediate node is a node introduced by the binarization.

A different estimate is given by Angelov and Ljunglöf (2014), who succeed in parsing longer sentences and providing an A* estimate, which is guaranteed to find the best derivation.

6.3.2 Non-projective dependency conversion

Hall and Nivre (2008), Versley (2014), and Fernández-González and Martins (2015) apply a reversible dependency conversion to the Tiger treebank, and use a non-projective dependency parser to parse with the converted treebank. The method has the advantage of being fast due to the greedy nature of the arc-eager transition-based dependency parser that is employed. The parser copes with non-projectivity by reordering tokens during parsing. Experiments are reported on the full Tiger treebank without length restrictions.

6.3.3 Reducing fan-out

The most direct way of reducing the complexity of LCFRS parsing is to reduce the fan-out of the grammar.

Maier *et al.* (2012) introduces a linguistically motivated reduction of the fan-outs of the Negra and Penn treebanks to fan-out 2 (up to a single gap per constituent). This enables parsing of sentences of up to length 40.

Nederhof and Vogler (2014) introduce a method of synchronous parsing with an LCFRS and a definite clause grammar. A parameter allows the fan-out (and thus parsing complexity) of the LCFRS to be reduced. Experiments are reported on sentences of up to 30 words on a small section of the Tiger treebank.

6.3.4 Coarse-to-fine pruning

We will focus on coarse-to-fine pruning, introduced in Charniak *et al.* (2006) and applied to discontinuous parsing by van Cranenburgh (2012a), who reports parsing results on the Negra treebank without length restrictions. Compared to the previous methods, this method does not change the grammar, but adds several new grammars to be used as preprocessing steps. Compared to the outside estimates, this method exploits sentence-specific information, since pruning information is collected during parsing with the coarser grammars.

Pauls and Klein (2009) present a comparison of coarse-to-fine and (hierarchical A*) outside estimates, and conclude that except when near-optimality is required, coarse-to-fine is more effective as it prunes a larger number of unlikely constituents.

A similar observation is obtained from a comparison of the discontinuous coarse-to-fine method and the outside estimates of Angelov and Ljunglöf (2014): coarse-to-fine is faster with longer sentences (30 words and up), at the cost of not always producing the most likely derivation (Ljunglöf, personal communication).

6.4 *Coarse-to-fine pipeline*

In order to tame the complexity of LCFRS and DOP, we apply coarse-to-fine pruning. Different grammars are used in the sequel, each being an overgenerating approximation of the next. That is, a coarse grammar will generate a larger set of constituents than a fine grammar. Parsing with a coarser grammar is more efficient, and all constituents which can be ruled out as improbable with a coarser grammar can be discarded as candidates when parsing with the next grammar. A constituent is ruled improbable if it does not appear in the k-best derivations of a parse forest. We use the same setup as in van Cranenburgh (2012a); namely, we parse in three stages, using three different grammars:

1. Split-PCFG: A CFG approximation of the discontinuous treebank grammar; rewrites spans of discontinuous constituents independently.
2. PLCFRS: The discontinuous treebank grammar; rewrites discontinuous constituents in a single operation. A discontinuous span $X_n\langle x_1, \ldots, x_n\rangle$ is added to the chart only if all of $X_n^{*m}\langle x_m\rangle$ with $1 \leqslant m \leqslant n$ are part of the k-best derivations of the chart of the previous stage.
3. Disco-DOP: The discontinuous DOP grammar; uses tree fragments instead of individual productions from the treebank. A discontinuous span $X_n\langle x_1, \ldots, x_n\rangle$ is added to the chart only if $X_n\langle x_1, \ldots, x_n\rangle$ is part of the k-best derivations of the chart of the previous stage, or if X_n is an intermediate symbol introduced by the TSG compression.

The first stage is necessary because without pruning, the PLCFRS generates too many discontinuous spans, the majority of which are improbable or not even part of a complete derivation. The second stage is not necessary for efficiency but gives slightly better accuracy on discontinuous constituents.

For example, while parsing the sentence "Wake your friend up," the discontinuous VP "Wake … up" may be produced in the PLCFRS stage. Before allowing this constituent to enter into the agenda and the chart, the chart of the previous stage is consulted to see if the two discontinuous components "Wake" and "up" were part of the k-best derivations. In the DOP stage, multiple elementary trees may be headed by this discontinuous constituent, and again they are only allowed on the chart if the previous stage produced the constituent as part of its k-best derivations.

The initial values for k are 10,000 and 50 for the PLCFRS and DOP grammar respectively. These values are chosen to be able to directly compare the new approach with the results in van Cranenburgh (2012a). However, experimenting with a higher value for k for the DOP stage has shown to yield improved performance. In other coarse-to-fine work the pruning criterion is based on a posterior threshold (e.g., Charniak *et al.* 2006; Bansal and Klein 2010); the k-best approach has the advantage that it does not require the computation of inside and outside probabilities.

For the initial PCFG stage, we apply beam search as in Collins (1999). The highest scoring item in each cell is tracked and only items up to 10,000 times less probable are allowed to enter the chart.

Experiments and results are described in Sections 8–9.

7 DISCONTINUITY WITHOUT LCFRS

The idea up to now has been to generate discontinuous constituents using formal rewrite operations of LCFRS. It should be noted, however, that the PCFG approximation used in the pruning stage reproduces discontinuities using information derived from the non-terminal labels. Instead of using this technique only as a crutch for pruning, it can also be combined with the use of fragments to obtain a pipeline that runs in cubic time. While the CFG approximation increases the independence assumptions for discontinuous constituents, the use of large fragments in the DOP approach can mitigate this increase. To create the CFG approximation of the discontinuous treebank grammar, the treebank is transformed by splitting discontinuous constituents into several non-terminal nodes (as explained in Section 4.1), after which grammar productions are extracted. This last step can also be replaced with fragment extraction to obtain a DOP grammar from the transformed treebank. We shall refer to this alternative approach as 'Split-2DOP.' The coarse-to-fine pipeline is now as follows:

1. Split-PCFG: A treebank grammar based on the CFG approximation of discontinuous constituents; rewrites spans of discontinuous constituents independently.
2. Split-2DOP grammar: tree fragments based on the same transformed treebank as above.

Since every discontinuous non-terminal is split up into a new non-terminal for each of its spans, the independence assumptions for that non-terminal in a probabilistic grammar are increased. While this representation is not sufficient to express the full range of nested discontinuous configurations, it appears adequate for the linguistic phenomena in the treebanks used in this work, since their trees can be unambiguously transformed back and forth into this representation. Moreover, the machinery of Data-Oriented Parsing mitigates the increase in independence assumptions through the use of large fragments. We can therefore parse using a DOP model with a context-free grammar as the symbolic backbone, and still recover discontinuous constituents.

8 EXPERIMENTAL SETUP

In this section we describe the experimental setup for benchmarking our discontinuous Double-DOP implementations on several discontinuous treebanks.

8.1 *Treebanks and preprocessing*

We evaluate on three languages: for German, we use the Negra (Skut *et al.* 1997) and Tiger (Brants *et al.* 2002) treebanks; for English, we use a discontinuous version of the Penn treebank (Evang and Kallmeyer 2011); and for Dutch, we use the Lassy (Van Noord 2009) and CGN (van der Wouden *et al.* 2002) treebanks; cf. Table 1. Negra and Tiger contain discontinuous annotations by design, as a strategy to cope with the relatively free word order of German. The discontinuous Penn treebank consists of the WSJ section in which traces have

Table 1: The discontinuous treebanks used in the experiments and the number of sentences used for development, training, and testing

Treebank	train (sentences)	dev (sentences)	test (sentences)
GERMAN			
Negra	18,602	1000	1000
	(#1–18,602)	(#19,603–20,602)	(#18,603–19,602)
Tiger	40,379 / 45,427	5048	5047
ENGLISH			
PTB: WSJ	39,832	1346	2416
DUTCH			
Lassy small	52,157	6520	6523
CGN	70,277	2000	2000

been converted to discontinuous constituents; we use the version used in Evang and Kallmeyer (2011, Sections 5.1–5.2) without restrictions on the transformations. The Lassy treebank is referred to as a dependency treebank but when discontinuity is allowed it can be directly interpreted as a constituency treebank. The *Corpus Gesproken Nederlands* (CGN, Spoken Dutch Corpus; van der Wouden *et al.* 2002) is a Dutch spoken language corpus with the same syntactic annotations. We use the syntactically annotated sentences from the Netherlands (i.e., without the Flemish part) of up to 100 tokens. The train-dev-test splits we employ are as commonly used for the Penn treebank: sec. 2–21, sec. 24, sec. 23, respectively. For Negra we use the one defined in Dubey and Keller (2003). For Tiger we follow Hall and Nivre (2008) who define sections 0–9 where sentence i belongs to section i mod 10, sec. 0 is used as test, sec. 1 as development, and 2–9 as training. When parsing the Tiger test set, the development set is added to the training set as well; while this is not customary, it ensures the results are comparable with Hall and Nivre (2008).

The same split is applied to the CGN treebank but with a single training set. For Lassy the split is our own.[8]

[8] The Lassy split derives from 80–10–10 partitions of the canonically ordered sentence IDs in each subcorpus (viz. `dpc`, `WR`, `WS`, and `wiki`). Canonically ordered refers to a 'version sort' where an identifier such as '2.12.a' is treated as a tuple of three elements compared consecutively.

For purposes of training we apply heuristics for head assignment (Klein and Manning 2003) and binarize the trees in the training sets head-outward with $h = 1$, $v = 1$ markovization; i.e., n-ary nodes are factored into nodes specifying an immediate sibling and parent. Note that for LCFRS, a binarization may increase the fan-out, and thus the complexity of parsing. It is possible to select the binarization in such a way as to minimize this complexity (Gildea 2010). However, experiments show that this increase in fan-out does not actually occur, regardless of the binarization strategy (van Cranenburgh 2012a). Head-outward means that constituents are binarized in a right-factored manner up until the head child, after which the rest of the binarization continues in a left-factored manner.

We add fan-out markers to guarantee unique fan-outs for non-terminal labels, e.g., $\{\mathrm{VP}, \mathrm{VP}_2, \mathrm{VP}_3, \ldots\}$, which are removed again for evaluation.

For the Dutch and German treebanks, punctuation is not part of the syntactic annotations. This causes spurious discontinuities, as the punctuation interrupts the constituents dominating its surrounding tokens. Additionally, punctuation provides a signal for constituent boundaries, and it is useful to incorporate it as part of the rest of the phrase structures. We use the method described in van Cranenburgh (2012a): punctuation is attached to the highest constituent that contains a neighbor to its right. With this strategy there is no increase in the amount of discontinuity with respect to a version of the treebank with punctuation removed. The CGN treebank contains spoken language phenomena, including disfluencies such as interjections and repeated words. In preprocessing, we treat these as if they were punctuation tokens; i.e., they are moved to an appropriate constituent (as defined above) and are ignored in the evaluation.

The complexity of parsing with a binarized LCFRS is $O(n^{3\varphi})$ with φ the highest fan-out of the non-terminals in the grammar (Seki *et al.* 1991). For a given grammar, it is possible to give a tighter upper bound on the complexity of parsing. Given the unique fan-outs of non-terminals in a grammar, the number of operations it takes to apply a production is the sum of the fan-outs in the production (Gildea 2010):

$$c(p) = \varphi(A) + \sum_{i=1}^{r} \varphi(B_i)$$

The complexity of parsing with a grammar is then the maximum value of this measure for productions in the grammar. In our experiments we find a worst-case time complexity of $O(n^9)$ for parsing with the DOP grammars extracted from Negra and WSJ. The following sentence from Negra contributes a grammar production with complexity 9. The production is from the VP of *vorgeworfen*; bracketed words are from other constituents, indicating the discontinuities:

(2) Den Stadtteilparlamentariern [ist] immer wieder ["Kirchturmpolitik"]
The district-MPs have always again "parochialism"
vorgeworfen [worden], weil sie nicht über die Grenzen des
accused been, because they not beyond the boundaries of-the
Ortsbezirks hinausgucken würden.
local-district look-out would.
'Time and again, the district MPs have been accused of "parochialism" because they would not look out beyond the boundaries of the local district.'

The complexities for Tiger and Lassy are $O(n^{10})$ and $O(n^{12})$ respectively, due to a handful of anomalous sentences; by discarding these sentences, a grammar with a complexity of $O(n^9)$ can be obtained with no or negligible effect on accuracy.

8.2 *Unknown words*

In initial experiments the parser is trained and evaluated on gold standard part-of-speech tags, as in previous experiments on discontinuous parsing. Later we show results when tags are assigned automatically with a simple unknown word model, based on the Stanford parser (Klein and Manning 2003). An open class threshold σ determines which tags are considered open class tags; tags that rewrite more than σ words are considered open class tags, and words they rewrite are open class words. Open class words in the training set that do not occur more than 4 times are replaced with signatures based on a list of features; words in the test set which are not part of the known words from the training set are replaced with similar signatures. The features are defined in the Stanford parser as *Model 4*, which is relatively language independent; cf. Table 2 for the list of features.[9] Signatures are formed by concatenating the names of features that apply

[9] This table is based on code from the Stanford parser (release 2014-08-27), specifically the method `getSignature4` in the file `EnglishUnknownWord-Model.java`.

Feature	Description
AC	All capital letters
SC	Initial capital, first word in sentence
C	Initial capital, other position
L, U	Has lower / upper case letter
S	No letters
N, n	All digits / one or more digits
H, P, C	Has dash / period / comma
x	Last character if letter and length > 3

Table 2:
Unknown word features, Stanford parser *Model 4*

to a word; e.g., 'forty-two' gives `_UNK-L-H-o`. A probability mass ϵ is assigned for combinations of known open class words with unseen tags. We use $\epsilon = 0.01$. We tuned σ on each training set to ensure that no closed class words are identified as open class words; for English and German we use $\sigma = 150$, and we use $\sigma = 100$ for Dutch.

8.3 *Function tags*

We investigated two methods of having the parser produce function tags in addition to the usual phrase labels. The first method is to train a separate discriminative classifier that adds function tags to parse trees in a post-processing step. This approach is introduced in Blaheta and Charniak (2000). We employed their feature set.

Another approach is to simply append the function tags to the non-terminal labels, resulting in, e.g., NP-SBJ and NP-OBJ for subject and object noun phrases. While this approach introduces sparsity and may affect the performance without function tags, we found this approach to perform best and therefore report results with this approach. Gabbard *et al.* (2006) and Fraser *et al.* (2013) use this approach as well. Compared to the classifier approach, it does not require any tuning, and the resulting model is fully generative. We apply this to the Tiger, WSJ, and Lassy treebanks.

The Penn treebank differs from the German and Dutch treebanks with respect to function tags. The Penn treebank only has function tags on selected non-terminals (never on preterminals) and each non-terminal may have several function tags from four possible categories; whereas the German and Dutch treebanks have a single function tag on most non-terminals. The tag set also differs considerably: the Penn treebank has 20 function tags, Lassy has 31, and Tiger has 43.

8.4 *Treebank refinements*

We apply a set of manual treebank refinements based on previous work. In order to compare the results on Negra with previous work, we do not apply the state splits when working with gold standard POS tags.

For Dutch and German we split the POS tags for the sentence-ending punctuation '.!?'. For all treebanks we add the feature 'year' to the preterminal label of tokens with numbers in the range 1900–2040, and replace the token with 1970. Other numbers are replaced with 000.

8.4.1 Tiger

For Tiger we apply the refinements described in Fraser *et al.* (2013). Since the Negra treebank is only partially annotated with morphological information, we do not apply these refinements to that treebank.

8.4.2 WSJ

We follow the treebank refinements of Klein and Manning (2003) for the Wall Street Journal section of the Penn treebank.

8.4.3 Lassy

The Lassy treebank contains fine-grained part-of-speech tags with morphological features. It is possible to use the full part-of-speech tags as the preterminal labels, but this introduces sparsity. We select a subset of features to add to the preterminal labels:

- nouns: proper/regular;
- verbs: auxiliary/main, finite/infinite;
- conjunctions: coordinating/subordinating;
- pronouns: personal/demonstrative;
- pre- vs. postposition.

Additionally, we percolate the feature identifying finite and infinite verbs to the parents and grandparents of the verb.

For multi-word units (MWU), we append the label of its head child. This helps distinguish MWUs as being nominal, verbal, prepositional, or otherwise.

The last two transformations are based on those for Tiger. Unary NPs are added for single nouns and pronouns in sentential, prepositional and infinitival constituents. For conjuncts, the function tag of the parent is copied. Both transformations can be reversed.

Since the CGN treebank uses a different syntax for the fine-grained POS tags, we do not apply these refinements to that treebank.

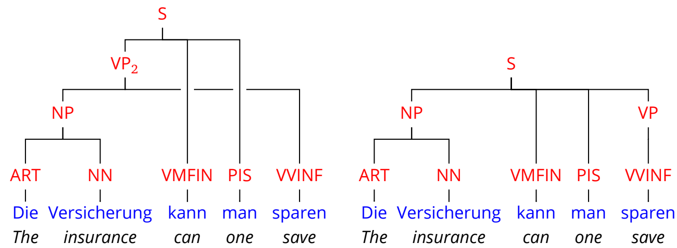
8.5 *Metrics*

We employ the exact match and Parseval measures (Black *et al.* 1992) as evaluation metrics. Both are based on bracketings that identify the label and yield of each constituent. The exact match is the proportion of sentences in which all labelled bracketings are correct. The Parseval measures consist of the precision, recall, and F-measure of the correct labelled bracketings averaged across the treebank. Since the POS accuracy is crucial to the performance of a parser and neither of the previous metrics reflect it, we also report the proportion of correct POS tags.

We use the evaluation parameters typically used with `EVALB` on the Penn treebank. Namely, the root node and punctuation are not counted towards the score (similar to `COLLINS.prm`,[10] except that we discount all punctuation, including brackets). Counting the root node as a constituent should not be done because it is not part of the corpus annotation and the parser is able to generate it without doing any work; when the root node is counted it inflates the F-score by several percentage points. Punctuation should be ignored because in the original annotation of the Dutch and German treebanks, punctuation is attached directly under the root node instead of as part of constituents. Punctuation can be re-attached using heuristics for the purposes of parsing, but evaluation should not be affected by this.

It is not possible to directly compare evaluation results from discontinuous parsing to existing state-of-the-art parsers that do not produce discontinuous constituents, since parses without discontinuous constituents contain a different set of bracketings; cf. Figure 15, which compares discontinuous bracketings to the bracketings extracted from a tree in which discontinuity has been resolved by attaching non-head siblings higher in the tree, as used in work on parsing Negra.

[10] This is part of the EVALB software, cf. `http://nlp.cs.nyu.edu/evalb/`



$\langle \mathbf{S}, \{1 \ldots 5\} \rangle$ $\langle \mathbf{S}, \{1 \ldots 5\} \rangle$
$\langle \mathbf{VP}_2, \{1, 2, 5\} \rangle$ $\langle \mathbf{VP}, \{5\} \rangle$
$\langle \mathbf{NP}, \{1, 2\} \rangle$ $\langle \mathbf{NP}, \{1, 2\} \rangle$

Figure 15: Bracketings from a tree with and without discontinuous constituents

Compared to an evaluation of bracketings without discontinuous constituents, an evaluation including discontinuous bracketings is more stringent. This is because bracketings are scored in an all-or-nothing manner, and a discontinuous bracketing includes non-local elements that would be scored separately when discontinuity is removed in a preprocessing step.

For function tags we use two metrics:

1. The non-null metric of Blaheta and Charniak (2000), which is the F-score of function tags on all correctly parsed bracketings. Since the German and Dutch treebanks include function tags on pre-terminals, we also include function tags on correctly tagged words in this metric.
2. A combined F-measure on bracketings of the form $\langle C, F, \text{span} \rangle$, where C is a syntactic category and F a function tag.

9 EVALUATION

This section presents an evaluation on three languages, and with respect to the use of function tags, tree fragments, pruning, and probabilities.

9.1 *Main results on three languages*

Table 3 lists the results for discontinuous parsing of three Germanic languages, with unknown word models. The cited works by Kallmeyer and Maier (2013) and Evang and Kallmeyer (2011) also use LCFRS for discontinuity but employ a treebank grammar with relative frequencies of productions. Hall and Nivre (2008), Versley (2014), and Fernández-González and Martins (2015) use a conversion to dependencies discussed in Section 6.3.2. For English and German our results improve upon the best known discontinuous constituency parsing results. The new system achieves a 16% relative error reduction over the previous best result for discontinuous parsing on sentences of size $\leqslant 40$ in the Negra test set. In terms of efficiency, the Disco-2DOP model is more than three times as fast as the DOP reduction, taking about three hours instead of ten on a single core. The grammar is also more compact: the Disco-2DOP grammar is only a third the size of that of the DOP reduction, at 6 MB versus 18 MB compressed size.

Table 3 also includes results from van Cranenburgh and Bod (2013) who do not add function tags to non-terminal labels nor apply the extensive treebank refinements described in Sections 8.3–8.4. Although the refinements and some of the function tags would be expected to improve performance, the rest of the function tags increase sparsity and consequently the resulting F-scores are slightly lower; but this tradeoff seems to be justified in order to get parse trees with function tags. The results on CGN show a surprisingly high exact match score. This is due to a large number of interjection utterances, e.g., "uhm."; since such sentences only consist of a root node and POS tags, the bracketing F_1-score is not affected by this.

9.2 *Function tags*

Table 4 reports an evaluation including function tags. For these three treebanks, the models reproduce most of the information in the original treebank. The following parts are not yet incorporated. The German and Dutch treebanks contain additional lexical information consisting of lemmas and morphological features. These could be added to the non-terminal labels of the model or obtained from an external POS tagger. Lastly, some non-terminals have multiple parents; these occur in the German and Dutch treebanks and are referred to as secondary edges.

9.3 *All-fragments vs. recurring fragments*

The original Disco-DOP model (van Cranenburgh *et al.* 2011) is based on an all-fragments model, while Disco-2DOP is based on recurring

Table 3: Discontinuous parsing of three Germanic languages. POS is the part-of-speech tagging accuracy; F_1 is the labelled bracketing F_1-score; EX is the exact match score. Results marked with * use gold standard POS tags; those marked with † do not discount the root node and punctuation. NB: Kallmeyer and Maier (2013) and Evang and Kallmeyer (2011) use a different test set and length restriction. 'vanCraBod2013' refers to van Cranenburgh and Bod (2013), and 'FeMa2015' to Fernández-González and Martins (2015)

		DEV			TEST		
Treebank and parser	$\|w\|$	POS	F_1	EX	POS	F_1	EX
GERMAN							
Negra							
van Cranenburgh (2012a)*	$\leqslant 40$	100	74.3	34.3	100	72.3	33.2
Kallmeyer and Maier (2013)*†	$\leqslant 30$				100	75.8	
this work, Disco-2DOP*	$\leqslant 40$	100	**77.7**	**41.5**	100	**76.8**	**40.5**
this work, Disco-2DOP	$\leqslant 40$	96.7	76.4	39.2	96.3	74.8	38.7
Tiger							
Hall and Nivre (2008)	$\leqslant 40$				97.0	75.3	32.6
Versley (2014)	$\leqslant 40$				100	74.2	37.3
FeMa2015	$\leqslant 40$					**82.6**	**45.9**
vanCraBod2013, Disco-2DOP	$\leqslant 40$	97.6	78.7	40.5	97.6	78.8	40.8
this work, Disco-2DOP	$\leqslant 40$	96.6	78.3	40.2	96.1	78.2	40.0
this work, Split-2DOP	$\leqslant 40$	96.6	78.1	39.2	96.2	78.1	39.0
ENGLISH							
WSJ							
Evang and Kallmeyer (2011)*†	< 25				100	79.0	
vanCraBod2013, Disco-2DOP	$\leqslant 40$	96.0	85.2	28.0	96.6	85.6	31.3
this work, Disco-2DOP	$\leqslant 40$	96.1	**86.9**	**29.5**	96.7	**87.0**	**34.4**
this work, Split-2DOP	$\leqslant 40$	96.1	86.7	**29.5**	96.7	87.0	33.9
DUTCH							
Lassy							
vanCraBod2013, Disco-2DOP	$\leqslant 40$	94.1	**79.0**	**37.4**	94.6	**77.0**	**35.2**
this work, Disco-2DOP	$\leqslant 40$	96.7	78.3	36.2	96.3	76.6	34.0
this work, Split-2DOP	$\leqslant 40$	96.8	78.0	34.9	96.3	76.2	32.7
CGN							
this work, Disco-2DOP	$\leqslant 40$	96.7	**72.6**	**64.1**	96.7	**73.0**	**63.8**
this work, Split-2DOP	$\leqslant 40$	96.6	71.2	63.4	96.7	72.2	63.3

Language, treebank	phrase labels	function tags	combined
German, Tiger	78.2	93.5	68.1
English, WSJ	87.0	86.3	82.5
Dutch, Lassy	76.6	92.8	70.0

Table 4:
Evaluation of function tags on sentences $\leqslant 40$ words, test sets; F_1 scores as defined at the end of Section 8.5

fragments. Table 5 compares previous results of Disco-DOP to the new Disco-2DOP implementation. The second column shows the accuracy for different values of k, i.e., the number of coarse derivations that determine the allowed labelled spans for the fine stage. While increasing this value did not yield improvements using the DOP reduction, with Disco-2DOP there is a substantial improvement in performance, with $k = 5000$ yielding the best score among the handful of values tested. Figure 16 shows the average time spent in each stage using the latter model on WSJ. The average time to parse a sentence ($\leqslant 40$ words) for this grammar is 7.7 seconds. Efficiency could be improved significantly by improving the PCFG parser using better chart representations such as packed parse forests and bit vectors (Schmid 2004).

Model	k = 50 F_1	k = 5000 F_1
DOP reduction: disco-DOP	74.3	73.5
Double-DOP: disco-2DOP	76.3	**77.7**

Table 5:
Comparing F-scores for the DOP reduction (implicit fragments) with Double-DOP (explicit fragments) on the Negra development set with different amounts of pruning (higher k means less pruning); gold standard POS tags

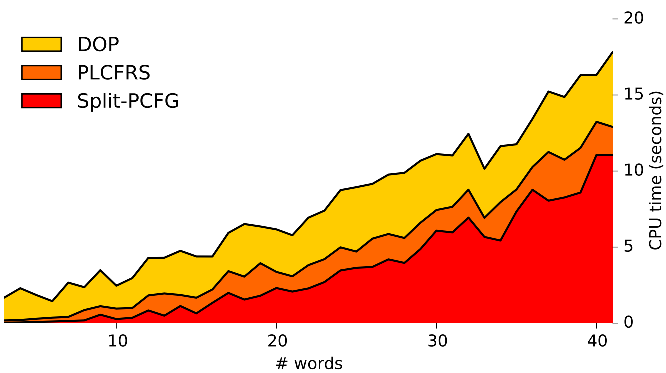


Figure 16:
Average time spent in each stage for sentences by length; disco-2DOP, WSJ development set

9.4 *Effects of pruning*

The effects of pruning can be further investigated by comparing different levels of pruning. We first parse the sentences in the Negra development set that are up to 30 words long with a PLCFRS treebank grammar, with $k = 10,000$ and without pruning. Out of 897 sentences, the Viterbi derivation is pruned on only 14 occasions, while the pruned version is about 300 times faster.

Table 6 shows results for different levels of pruning on sentences of all lengths. For sentences of all lengths it is not feasible to parse with the unpruned PLCFRS. However, we can compare the items in the parse forest after pruning and the best derivation to the gold tree from the treebank. From the various measures, it can be concluded that the pruning has a large effect on speed and the number of items in the resulting parse forest, while having only a small effect on the quality of the parse (forest).

Table 6: Results for different levels of pruning; mean over 1000 sentences

	(PCFG)	$k=100$	$k=1000$	$k=5000$	$k=10,000$
CPU time (seconds)	2.461	0.128	0.193	0.444	0.739
Number of items in chart	69,570.5	207.6	282.7	378.2	436.5
Percentage of gold standard items in chart	94.7	94.2	97.2	98.1	98.4
F_1 score	69.3	69.8	69.9	69.9	69.8

9.5 *Without LCFRS*

Table 3 shows that the Disco-2DOP and Split-2DOP techniques have comparable performance, demonstrating that the complexity of LCFRS parsing can be avoided. Table 7 shows the performance in each step of the coarse-to-fine pipelines, with and without LCFRS. Surprisingly, the use of a formalism that explicitly models discontinuity as an operation does not give any improvement over a simpler model in which discontinuities are only modeled probabilistically by encoding them into labels and fragments. This demonstrates that given the use of tree fragments, discontinuous rewriting through LCFRS comes at a high computational cost without a clear benefit over CFG.

Pipeline	F_1	EX%
Split-PCFG (no LCFRS, no TSG)	65.8	28.0
Split-PCFG ⇒ PLCFRS (no TSG)	65.9	28.6
Split-PCFG ⇒ PLCFRS ⇒ 2DOP	77.7	41.5
Split-PCFG ⇒ Split-2DOP (no LCFRS)	78.1	42.0

Table 7:
Parsing discontinuous constituents is possible without LCFRS (Negra development set, gold standard POS tags; results are for final stage)

9.6 *The role of probabilities*

From the results it is clear that a probabilistic tree-substitution grammar is able to provide much better results than a simple treebank grammar. However, it is not obvious whether the improvement is specifically due to the more fine-grained statistics (i.e., frequencies of more specific events), or generally because of the use of larger chunks. A serendipitous discovery during development of the parser provides insight into this: during an experiment, the frequencies of fragments were accidentally permuted and assigned to different fragments, but the resulting decrease in performance was surprisingly low, from 77.7 to 74.1 F_1 – suggesting that most of the improvement over the 65.9 F_1 score of the PLCFRS treebank grammar comes from memorizing larger chunks, as opposed to statistical reckoning.

9.7 *Previous work*

Earlier work on recovering empty categories and their antecedents in the Penn treebank (Johnson 2002; Levy and Manning 2004; Gabbard *et al.* 2006; Schmid 2006; Cai *et al.* 2011) has recovered nonlocal dependencies by producing the traces and co-indexation as in the original annotation. If the results include both traces and antecedents (which holds for all but the last work cited), the conversion to discontinuous constituents of Evang and Kallmeyer (2011) could be applied to obtain a discontinuous F-score. Since this would require access to the original parser output, we have not pursued this.

As explained in Section 8.5, it is not possible to directly compare the results to existing parsers that do not produce discontinuous constituents. However, the F-measures do give a rough measure, since the majority of constituents are not discontinuous.

For English, there is a result with 2DOP by Sangati and Zuidema (2011) with an F_1 score of 87.9. This difference can be attributed to the absence of discontinuous bracketings, as well as their use of the Maximum Constituents Parse instead of the Most Probable Parse; the former optimizes the F-measure instead of the exact match score. Shindo *et al.* (2012) achieve an F_1 score of 92.9 with a Bayesian TSG that uses symbol refinement through latent variables (i.e., automatic state splitting).

For German, the best results without discontinuity and no length restriction are F_1 scores of 84.2 for Negra (Petrov 2010) and 76.8 for Tiger (Fraser *et al.* 2013; note that this result employs a different train-dev-test split than the one in this work).

10 CONCLUSION

We have shown how to parse with discontinuous tree-substitution grammars and presented a practical implementation. We employ a fragment extraction method that finds recurring structures in treebanks efficiently, and supports discontinuous treebanks. This enables a data-oriented parsing implementation that employs a compact, efficient, and accurate model for discontinuous parsing in a generative model that improves upon previous results for this task.

Surprisingly, it turns out that the formal power of LCFRS is not necessary to describe discontinuity, since equivalent results can be obtained with a probabilistic tree-substitution grammar in which non-local relations are encoded in the non-terminal labels. In other words, it is feasible to produce discontinuous constituents without invoking mild context-sensitivity.

We have presented parsing results on three languages. Compared to previous work on statistical parsing, our models are linguistically richer. In addition to discontinuous constituents, our models also reproduce function tags from the treebank. While there have been previous results on reproducing non-local relations or function tags, this work reproduces both using models derived straightforwardly from treebanks, while exploiting ready-made treebank transformations for improved performance.

The source code of the parser used in this work is available at `https://github.com/andreasvc/disco-dop`.

ACKNOWLEDGMENTS

We are grateful to Kilian Evang for supplying the discontinuous Penn treebank, to the reviewers for detailed comments, and to Dave Carter and Adam Przepiórkowski for copy-editing suggestions.

This work is supported by the Computational Humanities Program of the Royal Netherlands Academy of Arts and Sciences, as part of The Riddle of Literary Quality.

REFERENCES

Krasimir ANGELOV and Peter LJUNGLÖF (2014), Fast statistical parsing with parallel multiple context-free grammars, in *Proceedings of EACL*, pp. 368–376, `http://aclweb.org/anthology/E14-1039`.

Mohit BANSAL and Dan KLEIN (2010), Simple, accurate parsing with an all-fragments grammar, in *Proceedings of ACL*, pp. 1098–1107, `http://aclweb.org/anthology/P10-1112`.

François BARTHÉLEMY, Pierre BOULLIER, Philippe DESCHAMP, and Éric DE LA CLERGERIE (2001), Guided parsing of range concatenation languages, in *Proceedings of ACL*, pp. 42–49, `http://aclweb.org/anthology/P01-1007`.

Shane BERGSMA, Matt POST, and David YAROWSKY (2012), Stylometric analysis of scientific articles, in *Proceedings of NAACL*, pp. 327–337, `http://aclweb.org/anthology/N12-1033`.

Ezra BLACK, John LAFFERTY, and Salim ROUKOS (1992), Development and evaluation of a broad-coverage probabilistic grammar of English-language computer manuals, in *Proceedings of ACL*, pp. 185–192, `http://aclweb.org/anthology/P92-1024`.

Don BLAHETA and Eugene CHARNIAK (2000), Assigning function tags to parsed text, in *Proceedings of NAACL*, pp. 234–240, `http://aclweb.org/anthology/A00-2031`.

Rens BOD (1992), A computational model of language performance: data-oriented parsing, in *Proceedings COLING*, pp. 855–859, `http://aclweb.org/anthology/C92-3126`.

Rens BOD (1995), The problem of computing the most probable tree in data-oriented parsing and stochastic tree grammars, in *Proceedings of EACL*, pp. 104–111, `http://aclweb.org/anthology/E95-1015`.

Rens BOD (2001), What is the minimal set of fragments that achieves maximal parse accuracy?, in *Proceedings of ACL*, pp. 69–76, `http://aclweb.org/anthology/P01-1010`.

Rens BOD, Remko SCHA, and Khalil SIMA'AN, editors (2003), *Data-Oriented Parsing*, The University of Chicago Press.

Pierre BOULLIER (1998), Proposal for a natural language processing syntactic backbone, Technical Report RR-3342, INRIA-Rocquencourt, Le Chesnay, France, `http://www.inria.fr/RRRT/RR-3342.html`.

Adriane BOYD (2007), Discontinuity revisited: An improved conversion to context-free representations, in *Proceedings of the Linguistic Annotation Workshop*, pp. 41–44, `http://aclweb.org/anthology/W07-1506`.

Sabine BRANTS, Stefanie DIPPER, Silvia HANSEN, Wolfgang LEZIUS, and George SMITH (2002), The Tiger treebank, in *Proceedings of the workshop on treebanks and linguistic theories*, pp. 24–41, `http://www.bultreebank.org/proceedings/paper03.pdf`.

Joan BRESNAN, Ronald M. KAPLAN, Stanley PETERS, and Annie ZAENEN (1982), Cross-serial dependencies in Dutch, *Linguistic Inquiry*, 13(4):613–635.

Shu CAI, David CHIANG, and Yoav GOLDBERG (2011), Language-independent parsing with empty elements, in *Proceedings of ACL-HLT*, pp. 212–216, `http://aclweb.org/anthology/P11-2037`.

Samy CHAMBI, Daniel LEMIRE, Owen KASER, and Robert GODIN (2015), Better bitmap performance with Roaring bitmaps, *Software: Practice and Experience*, ISSN 1097-024X, doi:10.1002/spe.2325, `http://arxiv.org/abs/1402.6407`, to appear.

Eugene CHARNIAK (1996), Tree-bank grammars, in *Proceedings of the National Conference on Artificial Intelligence*, pp. 1031–1036.

Eugene CHARNIAK, Mark JOHNSON, M. ELSNER, J. AUSTERWEIL, D. ELLIS, I. HAXTON, C. HILL, R. SHRIVATHS, J. MOORE, M. POZAR, *et al.* (2006), Multilevel coarse-to-fine PCFG parsing, in *Proceedings of NAACL-HLT*, pp. 168–175, `http://aclweb.org/anthology/N06-1022`.

David CHIANG (2000), Statistical parsing with an automatically-extracted tree adjoining grammar, in *Proceedings of ACL*, pp. 456–463, `http://aclweb.org/anthology/P00-1058`.

Noam CHOMSKY (1956), Three models for the description of language, *IRE Transactions on Information Theory*, 2(3):113–124.

Noam CHOMSKY (1965), *Aspects of the Theory of Syntax*, MIT press.

Trevor COHN, Phil BLUNSOM, and Sharon GOLDWATER (2010), Inducing tree-substitution grammars, *The Journal of Machine Learning Research*, 11(Nov):3053–3096.

Trevor COHN, Sharon GOLDWATER, and Phil BLUNSOM (2009), Inducing compact but accurate tree-substitution grammars, in *Proceedings of NAACL-HLT*, pp. 548–556, `http://aclweb.org/anthology/N09-1062`.

Michael COLLINS (1999), *Head-driven statistical models for natural language parsing*, Ph.D. thesis, University of Pennsylvania.

Peter DIENES and Amit DUBEY (2003), Deep syntactic processing by combining shallow methods, in *Proceedings of ACL*, pp. 431–438, http://aclweb.org/anthology/P03-1055.

Amit DUBEY and Frank KELLER (2003), Probabilistic parsing for German using sister-head dependencies, in *Proceedings of ACL*, pp. 96–103, http://aclweb.org/anthology/P03-1013.

Kilian EVANG and Laura KALLMEYER (2011), PLCFRS parsing of English discontinuous constituents, in *Proceedings of IWPT*, pp. 104–116, http://aclweb.org/anthology/W11-2913.

Daniel FERNÁNDEZ-GONZÁLEZ and André F. T. MARTINS (2015), Parsing as reduction, in *Proceedings of ACL*, pp. 1523–1533, http://aclweb.org/anthology/P15-1147.

Alexander FRASER, Helmut SCHMID, Richárd FARKAS, Renjing WANG, and Hinrich SCHÜTZE (2013), Knowledge sources for constituent parsing of German, a morphologically rich and less-configurational language, *Computational Linguistics*, 39(1):57–85, http://aclweb.org/anthology/J13-1005.

Ryan GABBARD, Mitchell MARCUS, and Seth KULICK (2006), Fully parsing the Penn treebank, in *Proceedings of NAACL-HLT*, pp. 184–191, http://aclweb.org/anthology/N06-1024.

Stuart GEMAN and Mark JOHNSON (2004), Probability and statistics in computational linguistics, a brief review, in Mark JOHNSON, Sanjeev P. KHUDANPUR, Mari OSTENDORF, and Roni ROSENFELD, editors, *Mathematical foundations of speech and language processing*, pp. 1–26, Springer.

Daniel GILDEA (2010), Optimal parsing strategies for linear context-free rewriting systems, in *Proceedings of NAACL-HLT*, pp. 769–776, http://aclweb.org/anthology/N10-1118.

Joshua GOODMAN (2003), Efficient parsing of DOP with PCFG-reductions, in Bod *et al.* (2003), pp. 125–146.

Spence GREEN, Marie-Catherine DE MARNEFFE, John BAUER, and Christopher D. MANNING (2011), Multiword expression identification with tree substitution grammars: A parsing tour de force with French, in *Proceedings of EMNLP*, pp. 725–735, http://aclweb.org/anthology/D11-1067.

Johan HALL and Joakim NIVRE (2008), Parsing discontinuous phrase structure with grammatical functions, in Bengt NORDSTRÖM and Aarne RANTA, editors, *Advances in Natural Language Processing*, volume 5221 of *Lecture Notes in Computer Science*, pp. 169–180, Springer, http://dx.doi.org/10.1007/978-3-540-85287-2_17.

Lars HOOGWEG (2003), Extending DOP with insertion, in Bod *et al.* (2003), pp. 317–335.

Yu-Yin HSU (2010), Comparing conversions of discontinuity in PCFG parsing, in *Proceedings of Treebanks and Linguistic Theories*, pp. 103–113, `http://hdl.handle.net/10062/15954`.

Liang HUANG and David CHIANG (2005), Better k-best parsing, in *Proceedings of IWPT*, pp. 53–64, NB corrected version on author homepage: `http://www.cis.upenn.edu/~lhuang3/huang-iwpt-correct.pdf`.

Marinus A.C. HUYBREGTS (1976), Overlapping dependencies in Dutch, *Utrecht Working Papers in Linguistics*, 1:24–65.

Mark JOHNSON (2002), A simple pattern-matching algorithm for recovering empty nodes and their antecedents, in *Proceedings of ACL*, pp. 136–143, `http://aclweb.org/anthology/P02-1018`.

Aravind K. JOSHI (1985), How much context sensitivity is necessary for characterizing structural descriptions: Tree adjoining grammars, in David R. DOWTY, Lauri KARTTUNEN, and Arnold M. ZWICKY, editors, *Natural language parsing: Psychological, computational and theoretical perspectives*, pp. 206–250, Cambridge University Press, New York.

Miriam KAESHAMMER and Vera DEMBERG (2012), German and English treebanks and lexica for tree-adjoining grammars, in *Proceedings of LREC*, pp. 1880–1887, `http://www.lrec-conf.org/proceedings/lrec2012/pdf/398_Paper.pdf`.

Laura KALLMEYER (2009), A declarative characterization of different types of multicomponent tree adjoining grammars, *Research on Language and Computation*, 7(1):55–99.

Laura KALLMEYER (2010), *Parsing Beyond Context-Free Grammars*, Cognitive Technologies, Springer.

Laura KALLMEYER and Wolfgang MAIER (2010), Data-driven parsing with probabilistic linear context-free rewriting systems, in *Proceedings of COLING*, pp. 537–545, `http://aclweb.org/anthology/C10-1061`.

Laura KALLMEYER and Wolfgang MAIER (2013), Data-driven parsing using probabilistic linear context-free rewriting systems, *Computational Linguistics*, 39(1):87–119, `http://aclweb.org/anthology/J13-1006`.

Laura KALLMEYER, Wolfgang MAIER, and Giorgio SATTA (2009), Synchronous rewriting in treebanks, in *Proceedings of IWPT*, `http://aclweb.org/anthology/W09-3810`.

Fred KARLSSON (2007), Constraints on multiple centre-embedding of clauses, *Journal of Linguistics*, 43(2):365–392.

Dan KLEIN and Christopher D. MANNING (2003), Accurate unlexicalized parsing, in *Proceedings of ACL*, volume 1, pp. 423–430, http://aclweb.org/anthology/P03-1054.

Marco KUHLMANN (2013), Mildly non-projective dependency grammar, *Computational Linguistics*, 39(2):355–387, http://aclweb.org/anthology/J13-2004.

Marco KUHLMANN and Giorgio SATTA (2009), Treebank grammar techniques for non-projective dependency parsing, in *Proceedings of EACL*, pp. 478–486, http://aclweb.org/anthology/E09-1055.

Roger LEVY (2005), *Probabilistic models of word order and syntactic discontinuity*, Ph.D. thesis, Stanford University.

Roger LEVY and Christopher D. MANNING (2004), Deep dependencies from context-free statistical parsers: correcting the surface dependency approximation, in *Proceedings of ACL*, pp. 327–334, http://aclweb.org/anthology/P04-1042.

Wolfgang MAIER, Miriam KAESHAMMER, Peter BAUMANN, and Sandra KÜBLER (2014), Discosuite – A parser test suite for German discontinuous structures, in *Proceedings of LREC*, http://www.lrec-conf.org/proceedings/lrec2014/pdf/230_Paper.pdf.

Wolfgang MAIER, Miriam KAESHAMMER, and Laura KALLMEYER (2012), PLCFRS parsing revisited: Restricting the fan-out to two, in *Proceedings of TAG*, volume 11, http://wolfgang-maier.net/pub/tagplus12.pdf.

Wolfgang MAIER and Timm LICHTE (2011), Characterizing discontinuity in constituent treebanks, in *Proceedings of Formal Grammar 2009*, pp. 167–182, Springer.

Wolfgang MAIER and Anders SØGAARD (2008), Treebanks and mild context-sensitivity, in *Proceedings of Formal Grammar 2008*, pp. 61–76.

James D. MCCAWLEY (1982), Parentheticals and discontinuous constituent structure, *Linguistic Inquiry*, 13(1):91–106, http://www.jstor.org/stable/4178261.

Mark-Jan NEDERHOF and Heiko VOGLER (2014), Hybrid grammars for discontinuous parsing, in *Proceedings of COLING*, pp. 1370–1381, http://aclweb.org/anthology/C14-1130.

Timothy J. O'DONNELL, Joshua B. TENENBAUM, and Noah D. GOODMAN (2009), Fragment grammars: Exploring computation and reuse in language, Technical Report MIT-CSAIL-TR-2009-013, MIT CSAIL, http://hdl.handle.net/1721.1/44963.

Almerindo E. OJEDA (1988), A linear precedence account of cross-serial dependencies, *Linguistics and Philosophy*, 11(4):457–492.

Adam PAULS and Dan KLEIN (2009), Hierarchical search for parsing, in *Proceedings of NAACL-HLT*, pp. 557–565, http://aclweb.org/anthology/N09-1063.

P. Stanley PETERS and R. W. RITCHIE (1973), On the generative power of transformational grammars, *Information Sciences*, 6:49–83, http://dx.doi.org/10.1016/0020-0255(73)90027-3.

Slav PETROV (2010), Products of random latent variable grammars, in *Proceedings of NAACL-HLT*, pp. 19–27, http://aclweb.org/anthology/N10-1003.

Kenneth L. PIKE (1943), Taxemes and immediate constituents, *Language*, 19(2):65–82, http://www.jstor.org/stable/409840.

Matt POST (2011), Judging grammaticality with tree substitution grammar derivations, in *Proceedings of the ACL-HLT 2011*, pp. 217–222, http://aclweb.org/anthology/P11-2038.

Matt POST and Daniel GILDEA (2009), Bayesian learning of a tree substitution grammar, in *Proceedings of the ACL-IJCNLP 2009 Conference, Short Papers*, pp. 45–48, http://aclweb.org/anthology/P09-2012.

Brian ROARK, Kristy HOLLINGSHEAD, and Nathan BODENSTAB (2012), Finite-state chart constraints for reduced complexity context-free parsing pipelines, *Computational Linguistics*, 38(4):719–753, http://aclweb.org/anthology/J12-4002.

Federico SANGATI and Willem ZUIDEMA (2011), Accurate parsing with compact tree-substitution grammars: Double-DOP, in *Proceedings of EMNLP*, pp. 84–95, http://aclweb.org/anthology/D11-1008.

Federico SANGATI, Willem ZUIDEMA, and Rens BOD (2010), Efficiently extract recurring tree fragments from large treebanks, in *Proceedings of LREC*, pp. 219–226, http://dare.uva.nl/record/371504.

Remko SCHA (1990), Language theory and language technology; competence and performance, in Q.A.M. DE KORT and G.L.J. LEERDAM, editors, *Computertoepassingen in de Neerlandistiek*, pp. 7–22, LVVN, Almere, the Netherlands, original title: Taaltheorie en taaltechnologie; competence en performance. English translation: http://iaaa.nl/rs/LeerdamE.html.

Yves SCHABES and Richard C. WATERS (1995), Tree insertion grammar: cubic-time, parsable formalism that lexicalizes context-free grammar without changing the trees produced, *Computational Linguistics*, 21(4):479–513, http://aclweb.org/anthology/J95-4002.

Helmut SCHMID (2004), Efficient parsing of highly ambiguous context-free grammars with bit vectors, in *Proceedings of COLING '04*, http://aclweb.org/anthology/C04-1024.

Helmut SCHMID (2006), Trace prediction and recovery with unlexicalized PCFGs and slash features, in *Proceedings of COLING-ACL*, pp. 177–184, `http://aclweb.org/anthology/P06-1023`.

William SCHULER, Samir ABDELRAHMAN, Tim MILLER, and Lane SCHWARTZ (2010), Broad-coverage parsing using human-like memory constraints, *Computational Linguistics*, 36(1):1–30, `http://aclweb.org/anthology/J10-1001`.

William SCHULER, David CHIANG, and Mark DRAS (2000), Multi-component TAG and notions of formal power, in *Proceedings of ACL*, pp. 448–455, `http://aclweb.org/anthology/P00-1057`.

Hiroyuki SEKI, Takahashi MATSUMURA, Mamoru FUJII, and Tadao KASAMI (1991), On multiple context-free grammars, *Theoretical Computer Science*, 88(2):191–229.

Stuart M. SHIEBER (1985), Evidence against the context-freeness of natural language, *Linguistics and Philosophy*, 8:333–343.

Hiroyuki SHINDO, Yusuke MIYAO, Akinori FUJINO, and Masaaki NAGATA (2012), Bayesian symbol-refined tree substitution grammars for syntactic parsing, in *Proceedings of ACL*, pp. 440–448, `http://aclweb.org/anthology/P12-1046`.

Khalil SIMA'AN (1997), Efficient Disambiguation by means of stochastic tree substitution grammars, in D. JONES and H. SOMERS, editors, *New Methods in Language Processing*, pp. 178–198, UCL Press, UK.

Wojciech SKUT, Brigitte KRENN, Thorsten BRANTS, and Hans USZKOREIT (1997), An annotation scheme for free word order languages, in *Proceedings of ANLP*, pp. 88–95, `http://aclweb.org/anthology/A97-1014`.

Ben SWANSON, Elif YAMANGIL, Eugene CHARNIAK, and Stuart SHIEBER (2013), A context free TAG variant, in *Proceedings of the ACL*, pp. 302–310, `http://aclweb.org/anthology/P13-1030`.

Benjamin SWANSON and Eugene CHARNIAK (2012), Native language detection with tree substitution grammars, in *Proceedings of ACL*, pp. 193–197, `http://aclweb.org/anthology/P12-2038`.

Heike TELLJOHANN, Erhard HINRICHS, and Sandra KÜBLER (2004), The Tüba-D/Z Treebank: Annotating German with a context-free backbone, in *Proceedings of LREC*, pp. 2229–2235, `http://www.lrec-conf.org/proceedings/lrec2004/pdf/135.pdf`.

Heike TELLJOHANN, Erhard W HINRICHS, Sandra KÜBLER, Heike ZINSMEISTER, and Kathrin BECK (2012), Stylebook for the Tübingen treebank of written German (TüBa-D/Z), technical report, Seminar für Sprachwissenschaft, Universität Tübingen, Germany, `http://www.sfs.uni-tuebingen.de/fileadmin/static/ascl/resources/tuebadz-stylebook-1201.pdf`.

Marten H. TRAUTWEIN (1995), *Computational pitfalls in tractable grammar formalisms*, Ph.D. thesis, University of Amsterdam, `http://www.illc.uva.nl/Research/Publications/Dissertations/DS-1995-15.text.ps.gz`.

Andreas VAN CRANENBURGH (2012a), Efficient parsing with linear context-free rewriting systems, in *Proceedings of EACL*, pp. 460–470, corrected version: `http://andreasvc.github.io/eacl2012corrected.pdf`.

Andreas VAN CRANENBURGH (2012b), Literary authorship attribution with phrase-structure fragments, in *Proceedings of CLFL*, pp. 59–63, revised version: `http://andreasvc.github.io/clfl2012.pdf`.

Andreas VAN CRANENBURGH (2014), Extraction of phrase-structure fragments with a linear average time tree kernel, *Computational Linguistics in the Netherlands Journal*, 4:3–16, ISSN 2211-4009, `http://www.clinjournal.org/sites/default/files/01-Cranenburgh-CLIN2014.pdf`.

Andreas VAN CRANENBURGH and Rens BOD (2013), Discontinuous parsing with an efficient and accurate DOP model, in *Proceedings of IWPT*, pp. 7–16, `http://www.illc.uva.nl/LaCo/CLS/papers/iwpt2013parser_final.pdf`.

Andreas VAN CRANENBURGH, Remko SCHA, and Federico SANGATI (2011), Discontinuous data-oriented parsing: A mildly context-sensitive all-fragments grammar, in *Proceedings of SPMRL*, pp. 34–44, `http://aclweb.org/anthology/W11-3805`.

Leonoor VAN DER BEEK, Gosse BOUMA, Robert MALOUF, and Gertjan VAN NOORD (2002), The Alpino dependency treebank, *Language and Computers*, 45(1):8–22.

Ton VAN DER WOUDEN, Heleen HOEKSTRA, Michael MOORTGAT, Bram RENMANS, and Ineke SCHUURMAN (2002), Syntactic analysis in the spoken Dutch corpus (CGN), in *Proceedings of LREC*, pp. 768–773, `http://www.lrec-conf.org/proceedings/lrec2002/pdf/71.pdf`.

Gertjan VAN NOORD (2009), Huge parsed corpora in Lassy, in *Proceedings of TLT7*, LOT, Groningen, The Netherlands.

Yannick VERSLEY (2014), Experiments with easy-first nonprojective constituent parsing, in *Proceedings of SPMRL-SANCL 2014*, pp. 39–53, `http://aclweb.org/anthology/W14-6104`.

K. VIJAY-SHANKER and David J. WEIR (1994), The equivalence of four extensions of context-free grammars, *Theory of Computing Systems*, 27(6):511–546.

K. VIJAY-SHANKER, David J. WEIR, and Aravind K. JOSHI (1987), Characterizing structural descriptions produced by various grammatical formalisms, in *Proceedings of ACL*, pp. 104–111, `http://aclweb.org/anthology/P87-1015`.

David J. WEIR (1988), *Characterizing mildly context-sensitive grammar formalisms*, Ph.D. thesis, University of Pennsylvania, http://repository.upenn.edu/dissertations/AAI8908403/.

Rulon S. WELLS (1947), Immediate constituents, *Language*, 23(2):81–117, http://www.jstor.org/stable/410382.

Fei XIA, Chung-Hye HAN, Martha PALMER, and Aravind JOSHI (2001), Automatically extracting and comparing lexicalized grammars for different languages, in *Proceedings of IJCAI*, pp. 1321–1330.

Elif YAMANGIL and Stuart SHIEBER (2012), Estimating compact yet rich tree insertion grammars, in *Proceedings of ACL*, pp. 110–114, http://aclweb.org/anthology/P12-2022.

Andreas ZOLLMANN and Khalil SIMA'AN (2005), A consistent and efficient estimator for data-oriented parsing, *Journal of Automata Languages and Combinatorics*, 10(2/3):367–388, http://staff.science.uva.nl/~simaan/D-Papers/JALCsubmit.pdf.

On different approaches to syntactic analysis into bi-lexical dependencies: An empirical comparison of direct, PCFG-based, and HPSG-based parsers

Angelina Ivanova[1], *Stephan Oepen*[1], *Rebecca Dridan*[1], *Dan Flickinger*[2], *Lilja Øvrelid*[1], *and Emanuele Lapponi*[1]
[1] University of Oslo, Department of Informatics
[2] Stanford University, Center for the Study of Language and Information

ABSTRACT

We compare three different approaches to parsing into syntactic, bi-lexical dependencies for English: a 'direct' data-driven dependency parser, a statistical phrase structure parser, and a hybrid, 'deep' grammar-driven parser. The analyses from the latter two are post-converted to bi-lexical dependencies. Through this 'reduction' of all three approaches to syntactic dependency parsers, we determine empirically what performance can be obtained for a common set of dependency types for English; in- and out-of-domain experimentation ranges over diverse text types. In doing so, we observe what trade-offs apply along three dimensions: accuracy, efficiency, and resilience to domain variation. Our results suggest that the hand-built grammar in one of our parsers helps in both accuracy and cross-domain parsing performance. When evaluated extrinsically in two downstream tasks – negation resolution and semantic dependency parsing – these accuracy gains do sometimes but not always translate into improved end-to-end performance.

Keywords: syntactic dependency parsing, domain variation

Journal of Language Modelling Vol 4, No 1 (2016), pp. 113–144

1 MOTIVATION

Bi-lexical dependencies, i.e. binary head–argument relations holding exclusively between lexical units, are widely considered an attractive target representation for syntactic analysis. At the same time, Cer *et al.* (2010) and Foster *et al.* (2011), inter alios, have demonstrated that higher dependency accuracies can sometimes be obtained by parsing into a phrase structure representation first, and then reducing parse trees into bi-lexical dependencies.[1] Thus, if one is willing to accept pure syntactic dependencies as a viable interface (and evaluation) representation, an experimental setup like the one of Cer *et al.* (2010) allows the exact experimental comparison of quite different parsing approaches.[2] Existing such studies to date have predominantly focused on purely data-driven (or statistical) parsers, i.e. systems where linguistic knowledge is exclusively acquired through supervised machine learning from annotated training data. For English, the venerable Wall Street Journal (WSJ) portion of the Penn Treebank (PTB; Marcus *et al.* 1993) has been the most common source of training data for phrase structure and dependency parsers.

Two recent developments make it possible to broaden the range of parsing approaches that can be assessed empirically on the task of deriving bi-lexical syntactic dependencies. Flickinger *et al.* (2012) make available another annotation layer over the same WSJ text, 'deep' syntacto-semantic analyses in the linguistic framework of Head-Driven Phrase Structure Grammar (HPSG; Pollard and Sag 1994; Flickinger 2000). This resource, dubbed DeepBank, is available since late 2012. For the type of HPSG analyses recorded in DeepBank, Zhang and Wang (2009) and Ivanova *et al.* (2012) define a reduction into bi-lexical syntactic dependencies, which they call Derivation Tree-Derived Dependencies (DT). Through application of the converter of Ivanova *et al.* (2012) to DeepBank, we can thus obtain a DT-annotated version of the standard WSJ text, to train and test a data-driven dependency and

[1] This conversion from one representation of syntax to another is lossy, in the sense of discarding constituency information, hence we consider it a reduction in linguistic detail.

[2] In contrast, much earlier work on cross-framework comparison involved post-processing parser outputs in form *and* content, into a target representation for which gold-standard annotations were available. In Section 2 below, we argue that such conversion inevitably introduces blur into the comparison.

phrase structure parser, respectively, and to compare parsing results to a hybrid, grammar-driven HPSG parser. Furthermore, we can draw on a set of additional corpora annotated in the same HPSG format (and thus amenable to conversion for both phrase structure and dependency parsing), instantiating a comparatively diverse range of domains and genres (Oepen *et al.* 2004). Adding this data to our setup for additional cross-domain testing, we seek to document not only what trade-offs apply in terms of dependency accuracy vs. parser efficiency, but also how these trade-offs are affected by domain and genre variation, and more generally how resilient the different approaches are to variation in parser inputs.

2 RELATED WORK

Comparing between parsers from different frameworks has long been an area of active interest, ranging from the original PARSEVAL design (Black *et al.* 1991), to evaluation against 'formalism-independent' dependency banks (King *et al.* 2003; Briscoe and Carroll 2006), to dedicated workshops (Bos *et al.* 2008). Grammatical Relations (GRs; Briscoe and Carroll 2006) have been the target of a number of benchmarks, but they require a heuristic mapping from 'native' parser outputs to the target representations for evaluation, which makes results hard to interpret. Clark and Curran (2007) established an upper bound by running the mapping process on gold-standard data, to put into perspective the mapped results from their CCG parser proper. When Miyao *et al.* (2007) carried out the same experiment for a number of different parsers, they showed that the loss of accuracy due to the mapping process can swamp any actual parser differences. As long as heuristic conversion is required before evaluation, cross-framework comparison inevitably includes a level of fuzziness. An alternative approach is possible when there is enough data available in a particular representation to train a statistical parser, and any necessary conversion between representations is deterministic and hence doesn't introduce the same fuzziness. One example of this approach is demonstrated by Cer *et al.* (2010), who used Stanford Dependencies (de Marneffe and Manning 2008) to evaluate a range of statistical parsers. Since there is a deterministic process for converting between PTB phrase structure trees and Stanford Dependencies, they were able to evaluate a large number of different parsers which can be trained on one or the other of these representations, using the standard PTB training and test data, without resorting to fuzzy mapping processes.

Fowler and Penn (2010) formally proved that a range of Combinatory Categorial Grammars (CCGs) are context-free. They trained the PCFG Berkeley parser on CCGBank, the CCG annotation of the PTB WSJ text (Hockenmaier and Steedman 2007), advancing the state of the art in terms of supertagging accuracy, PARSEVAL measures, and CCG dependency accuracy. They concluded that a specialized CCG parser is not necessarily more accurate than the general-purpose Berkeley parser; this study, however, fails to also take parser efficiency into account.

In related work for Dutch, Plank and van Noord (2010) suggest that, intuitively, one should expect that a grammar-driven system can be more resilient to domain shifts than a purely data-driven parser. In a contrastive study on parsing into Dutch syntactic dependencies, they substantiated this expectation by showing that their HPSG-based Alpino system performed better and was more resilient to domain variation than data-driven direct dependency parsers.

3 BACKGROUND: HPSG SYNTACTIC DEPENDENCIES

The dependency format we use is a deterministic conversion of HPSG derivation trees licensed by the English Resource Grammar (ERG; Flickinger 2000). Figure 1 of an ERG derivation tree, where labels of internal nodes name HPSG constructions (e.g. subject–head or head–complement: sb-hd_mc_c and hd-cmp_u_c, respectively; see Section 5.3.1 for more details on unary rules). Preterminals are labeled with fine-grained lexical categories – called ERG lexical types – that augment common parts of speech with additional information, for example argument structure or the distinction between count, mass, and proper nouns. In total, the ERG distinguishes about 250 construction types and 1000 lexical types.

ERG derivations are grounded in a formal theory of grammar that explicitly marks heads. For this reason mapping these trees onto projective bi-lexical dependencies is straightforward (Zhang and Wang 2009). Ivanova *et al.* (2012) coin the term DT for ERG Derivation Tree-Derived Dependencies, where they reduce the inventory of some 250

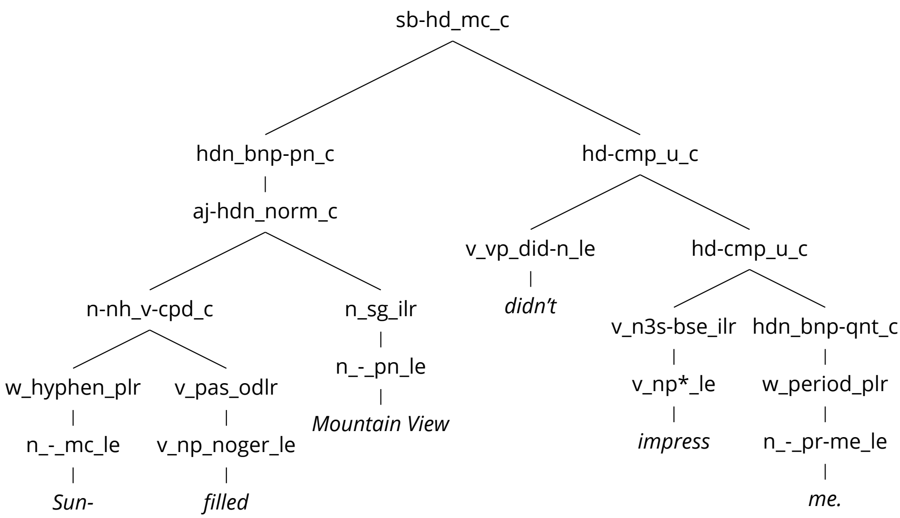


Figure 1: Sample HPSG derivation: construction identifiers label internal nodes, and lexical types the preterminals

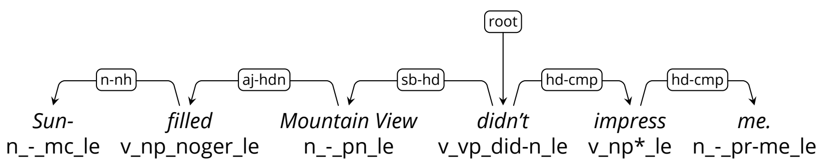


Figure 2: Sample DT bi-lexical dependencies: construction identifiers are generalized to major types (cutting at the first underscore)

ERG syntactic rules to 48 broad HPSG constructions.[3] The DT syntactic dependency tree for our running example is shown in Figure 2.

To better understand the nature of the DT scheme, Ivanova *et al.* (2012) offer a quantitative, structural comparison against two pre-existing dependency standards for English, viz. those from the CoNLL dependency parsing competitions (Nivre *et al.* 2007) and the 'basic' variant of Stanford Dependencies. They observe that the three dependency representations are broadly comparable in granularity and that there are substantial structural correspondences between the schemes.

[3] The ERG distinguishes main clause vs. subordinate subjects, for example, as seen in Figure 1. Ivanova *et al.* (2012) discard this and other grammar-internal contrasts by 'cutting' construction labels at the first underscore.

Measured as average Jaccard similarity over unlabeled dependencies, they observe the strongest correspondence between DT and CoNLL (at a Jaccard index of 0.49, compared to 0.32 for DT and Stanford, and 0.43 between CoNLL and Stanford).

Ivanova *et al.* (2013) complement the above discussed comparison of dependency schemes through an empirical assessment in terms of 'parsability', i.e. accuracy levels available for the different target representations when training and testing a range of state-of-the-art parsers on the same data sets. In their study, the dependency parser of Bohnet and Nivre (2012), henceforth B&N, consistently performs best for all schemes and output configurations. Furthermore, parsability differences between the representations are generally very small.

For a more exact comparison, we replicate their study and evaluate B&N for all three schemes when trained and tested on the same subset of PTB WSJ sentences that are available in DeepBank.[4] The results in Table 1 show that there are no interesting differences in performance of the Bohnet and Nivre (2012) parser across the DT, CoNLL, and Stanford Basic dependency schemes.

Table 1:
Parsability of three dependency schemes, measured as labeled attachment score (LAS) and unlabeled attachment score (UAS)

	LAS	UAS
CoNLL	90.53	93.56
Stanford	90.43	92.87
DT	90.48	92.77

Based on the observations from the above comparisons, we conjecture that DT is as suitable a target representation for parser comparison as any of the others. Furthermore, two linguistic factors add to the attractiveness of DT for our study: it is defined in terms of a formal (and implemented) theory of grammar; and it makes available more fine-grained lexical categories, ERG lexical types, than is common in PTB-derived dependency banks.

[4] For compatibility with much previous work, and to level the playing field for all three schemes, we opt for a slightly different setup for this comparison than in (most) subsequent experiments: here, we apply PTB-style tokenization, coarse-grained PTB parts of speech, and exclude punctuation from scoring.

4 DATA

Below we describe the construction and characteristics of the data sets we use in our parsing experiments, highlighting some of the relevant differences to the more widely-known Penn Treebank format.

4.1 *DeepBank*

DeepBank annotations were created by combining the native ERG parser PET (Callmeier 2002), with a discriminant-based tree selection tool (Carter 1997; Oepen *et al.* 2004), thus making it possible for annotators to navigate the large space of possible analyses efficiently, identify and validate the intended reading, and record its full HPSG analysis in the treebank. Owing to this setup, DeepBank version 1.0, as used presently, lacks analyses for some 15 percent of the WSJ sentences, for which either the ERG parser failed to suggest a set of candidates (within certain bounds on time and memory usage), or the annotators found none of the available parses acceptable.[5] Furthermore, DeepBank annotations to date only comprise the first 21 sections of the PTB WSJ corpus. Following the splits suggested by the DeepBank developers, we train on Sections 0–19, use Section 20 for tuning, and test against Section 21 (abbreviated as WSJ below).[6]

4.2 *Cross-domain test data*

Another benefit of the DT target representation is the availability of comparatively large and diverse samples of additional test data. The ERG Redwoods Treebank (Oepen *et al.* 2004) is similar in genealogy and format to DeepBank, comprising corpora from various domains and genres. Although Redwoods counts a total of some 400,000 annotated tokens, we only draw on it for additional *testing* data. In other words, we do not attempt parser re-training or adaptation against this additional data, but rather test our WSJ-trained parsers on out-of-domain samples from Redwoods. We report on four such test corpora,

[5] Thus, limitations in the ERG and PET effectively lead to the exclusion of a non-trivial percentage of sentences from our training and testing corpora. We discuss methodological ramifications of this setup to our study in Section 13 below.

[6] To 'protect' Section 21 as unseen test data, also for the ERG parser, this final section in Version 1.0 of DeepBank was not exposed to its developers until the grammar and disambiguation model were finalized and frozen for this release.

Table 2:
Sentence, token, and type counts for data sets

		Sentences	Tokens	Types
DeepBank	Train	33,783	661,451	56,582
	Tune	1,721	34,063	8,964
	WSJ	1,414	27, 515	7,668
Redwoods	CB	608	11,653	3,588
	SC	864	13,696	4,925
	VM	993	7,281	1,007
	WS	520	8,701	2,974

viz. (a) a software advocacy essay, *The Cathedral and the Bazaar* (CB); (b) a subset of the SemCor portion of the Brown Corpus (SC; Francis and Kučera 1982); (c) a collection of transcribed, task-oriented spoken dialogues (VM; Wahlster 2000); and (d) part of the Wikipedia-derived WeScience Corpus (WS; Ytrestøl *et al.* 2009). Table 2 provides exact sentence, token, and type counts for these data sets.

4.3 *Tokenization conventions*

A relevant peculiarity of the DeepBank and Redwoods annotations in this context is the ERG approach to tokenization. Three aspects in Figure 1 deviate from the widely used PTB conventions: (a) hyphens (and slashes) introduce token boundaries; (b) whitespace in multiword lexical units (like *ad hoc*, *of course*, or *Mountain View*) does not force token boundaries; and (c) punctuation marks are attached as 'pseudo-affixes' to adjacent words, reflecting the rules of standard orthography. Adolphs *et al.* (2008) offer some linguistic arguments for this approach to tokenization, but for our purposes it suffices to note that these differences to PTB tokenization may in part counter-balance each other in terms of overall parsing difficulty, but they do increase the types-per-tokens ratio somewhat. This property of the DeepBank annotations, arguably, makes English look somewhat similar to languages with moderate inflectional morphology. To take advantage of the fine-grained ERG lexical categories, most of our experiments assume ERG tokenization. In two calibration experiments, however, we also investigate the effects of tokenization differences on our parser comparison.

5 PARSERS

This section describes our three parsers, including the alternate configurations we use, and details of how they are trained and run.

5.1 *PET: native HPSG parsing*

The parser most commonly used with the ERG is called PET (Callmeier 2002), a highly engineered chart parser for unification grammars. PET constructs a complete parse forest, using subsumption-based ambiguity factoring (Oepen and Carroll 2000), and then extracts from the forest n-best lists of complete analyses according to a discriminative parse ranking model (Zhang *et al.* 2007). For our experiments, we employ ERG version 1212, train the parse ranker on Sections 00–19 of DeepBank, and otherwise use the default configuration (which corresponds to the environment used by the DeepBank and Redwoods developers), which is optimized for accuracy. This parser, performing exact inference, we will call ERG_a.

In recent work, Dridan (2013) has augmented ERG parsing with lattice-based sequence labeling over lexical types and lexical rules. Pruning the parse chart prior to forest construction yields greatly improved efficiency at a moderate accuracy loss. We include the best-performing configuration of Dridan (2013) in our experiments, a variant henceforth referred to as ERG_e.

Unlike the other parsers in our study, PET internally operates over an ambiguous token lattice, and there is no easy interface to feed the parser pre-tokenized inputs. We approximate the effects of gold-standard tokenization by requesting from the parser a 2000-best list, which we filter for the top-ranked analysis whose leaves match the treebank tokenization. This approach is imperfect, as in some cases no token-compatible analysis may be on the n-best list, especially so in the ERG_e setup (where lexical items may have been pruned by the sequence labeling model). When this happens, we fall back to the top-ranked analysis and adjust our evaluation metrics to robustly deal with tokenization mismatches (see Section 6).

5.2 *B&N: direct dependency parsing*

The parser of Bohnet and Nivre (2012), henceforth B&N, is a transition-based *dependency parser* with joint tagger that implements global learning and a beam search for non-projective labeled dependency parsing. This parser consistently outperforms pipeline systems (such as the Malt and MST parsers) both in terms of tagging and parsing accuracy for typologically diverse languages such as Chinese, English, and German. We apply B&N mostly 'out-of-the-box', training on the DT conversion of DeepBank Sections 00–19, and running the parser with an increased beam size of 80.

5.3 *Berkeley: PCFG parsing*

The Berkeley parser (Petrov *et al.* 2006), henceforth just Berkeley, is a generative, unlexicalized *phrase structure* parser that automatically derives a smoothed latent-variable PCFG from the treebank and refines the grammar by a split–merge procedure. The parser achieves state-of-the-art performance on various standard benchmarks.

Formally, the HPSG analyses in the DeepBank and Redwoods treebanks transcend the class of context-free grammars. Nevertheless, one can pragmatically look at an ERG derivation as if it were a context-free phrase structure tree. On this view, standard, off-the-shelf PCFG parsing techniques are applicable to the ERG treebanks. Zhang and Krieger (2011) explore this space experimentally, combining the ERG, Redwoods (but not DeepBank), and massive collections of automatically parsed text. Their study, however, does not consider parser efficiency.[7] In contrast, our goal is to reflect on practical trade-offs along multiple dimensions. We therefore focus on Berkeley, as one of the currently best-performing (and relatively efficient) PCFG engines. We train the parser on the derivation trees and then, for comparison to the other parsers in terms of DT dependency accuracy, we apply the converter of Ivanova *et al.* (2012) to Berkeley outputs. For technical reasons, however, the optional mapping from ERG to PTB tokenization is not applicable in this setup, and hence our experiments involving Berkeley are limited to ERG tokens and fine-grained lexical categories.

5.3.1 Tuning

Table 3 summarizes a series of experiments, seeking to tune the Berkeley parser for maximum accuracy on our development set, DeepBank

[7] Their best PCFG results are only a few points F_1 below the full HPSG parser, using very large PCFGs and exact inference; in this set-up, parsing times in fact exceed those of the native HPSG parser.

Table 3: Tagging accuracy, PARSEVAL F_1, and dependency accuracy for Berkeley on WSJ development data

	Unary Rules Removed			
Labels	Long		Short	
Cycles	5	6	5	6
Gaps	3	3	**0**	**0**
TA	88.46	87.65	89.16	88.46
F_1	74.53	73.72	75.15	73.56
LAS	83.96	83.20	80.49	79.56
UAS	87.12	86.54	87.95	87.15

	Unary Rules Preserved					
Labels	Long		Short		Mixed	
Cycles	5	6	5	6	5	6
Gaps	2	5	**0**	**0**	11	19
TA	90.96	90.62	91.11	**91.62**	90.93	90.94
F_1	76.39	75.66	79.81	**80.33**	76.70	76.74
LAS	86.26	85.90	82.50	83.15	**86.72**	86.16
UAS	89.34	88.92	89.80	**90.34**	89.42	88.84

Section 20. Due to its ability to internally rewrite node labels, this parser should be expected to adapt well also to ERG derivations. Compared to the phrase structure annotations in the PTB, there are two structural differences evident in Figure 1. First, the inventories of phrasal and lexical labels are larger, at around 250 and 1000, respectively, compared to only about two dozen phrasal categories and 45 parts of speech in the PTB. Second, ERG derivations contain more unary (non-branching) rules, recording for example morphological variation or syntacto-semantic category changes.[8]

We experiment with preserving unary rules in ERG derivations or removing them (as they make no difference to the final DT analysis); we further run experiments using the native ('long') ERG construc-

[8] Examples of morphological rules in Figure 1 include v_pas_odlr and v_n3s-bse_ilr, for passive-participle and non-third person singular or base inflection, respectively. Also, there are two instances of bare noun phrase formation: hdn_bnp-pn_c and hdn_bnp-qnt_c.

tion identifiers, their generalizations to 'short' labels as used in DT, and a variant with long labels for unary and short ones for branching rules ('mixed'). We report results for training with five or six split–merge cycles, where fewer iterations generally show inferior accuracy, and larger values lead to more parse failures ('gaps' in Table 3). There are some noticeable trade-offs across tagging accuracy, dependency accuracy, and coverage, without a single best performer along all three dimensions. As our primary interest across parsers is dependency accuracy, we select the configuration with unary rules and long labels, trained with five split–merge cycles, which seems to afford near-premium LAS at near-perfect coverage.[9]

6 EVALUATION

Standard evaluation metrics in dependency parsing are labeled and unlabeled attachment scores (LAS, UAS; implemented by the CoNLL eval.pl scorer). These measure the percentage of tokens which are correctly attached to their head token and, for LAS, have the right dependency label. As assignment of lexical categories is a core part of syntactic analysis, we complement LAS and UAS with tagging accuracy scores (TA), where appropriate. However, in our work there are two complications to consider when using eval.pl. First, some of our parsers occasionally fail to return any analysis, notably Berkeley and ERG_e. For these inputs, our evaluation re-inserts the missing tokens in the parser output, padding with dummy 'placeholder' heads and dependency labels.

Second, a more difficult issue is caused by occasional tokenization mismatches in ERG parses, as discussed above. Since eval.pl identifies tokens by their position in the sentence, any difference of tokenization will lead to invalid results. One option would be to treat all system outputs with token mismatches as parse failures, but this over-penalizes, as potentially correct dependencies among corresponding tokens are also removed from the parser output. For this reason, we modify the evaluation of dependency accuracy to use character offsets, instead of consecutive identifiers, to encode token identities. This way, tokenization mismatches local to some sub-segment of the input will not 'throw

[9] A welcome side-effect of this choice is that we end up using native ERG derivations without modifications.

off' token correspondences in other parts of the string.[10] We will refer to this character-based variant of the standard CoNLL metrics as LAS_c and UAS_c.

7 IN-DOMAIN PARSING RESULTS

Our first cross-paradigm comparison of the three parsers is against the WSJ in-domain test data, as summarized in Table 4. There are substantive differences between parsers both in terms of coverage, speed, and accuracy. Berkeley fails to return an analysis for one input, whereas ERG_e cannot parse 13 sentences (close to one percent of the test set); just as the 44 inputs where parser output deviates in tokenization from the treebank, this is likely an effect of the lexical pruning applied in this setup. At an average of one second per input, Berkeley is the fastest of our parsers; ERG_a is exactly one order of magnitude slower. However, the lexical pruning of Dridan (2013) in ERG_e leads to a speed-up of almost a factor of six, making this variant of PET perform comparable to B&N. The strongest differences, however, we observe in tagging and dependency accuracies. The two data-driven parsers perform very similarly (at close to 93% TA and around 86.7% LAS); the two ERG parsers are comparable too, but at accuracy levels that are four to six points higher in both TA and LAS. Compared to ERG_a, the faster ERG_e variant performs very slightly worse – which likely reflects penalization for missing coverage and token mismatches – but it nevertheless delivers much higher accuracy than the data-driven parsers.

	Gaps	Time	TA_c	LAS_c	UAS_c
Berkeley	1+0	**1.0**	92.9	86.65	89.86
B&N	**0+0**	1.7	92.9	86.76	89.65
ERG_a	**0+0**	10	**97.8**	**92.87**	**93.95**
ERG_e	13+44	1.8	96.4	91.60	92.72

Table 4: Parse failures and token mismatches ('gaps'), efficiency, and tagging and dependency accuracy on WSJ

[10] Where tokenization is identical for the gold and system outputs, the score given by this generalized metric is exactly the same as that of eval.pl. Unless indicated otherwise, punctuation marks are included in scoring.

8 CROSS-DOMAIN PARSING RESULTS

To gauge the resilience of the different systems to domain and genre variation, we apply the same set of parsers – without re-training or other adaptation – to the additional Redwoods test data. Table 5 summarizes coverage and accuracy results across the four diverse samples. Again, Berkeley and B&N pattern alike, with Berkeley slightly ahead in terms of dependency accuracy, but penalized on two of the test sets for parse failures. LAS for the two data-driven parsers ranges between 74% and 81%, up to 12 points below their WSJ performance. Though large, accuracy drops on a similar scale have been observed repeatedly for purely statistical systems when moving out of the WSJ domain without adaptation (Gildea 2001; Nivre *et al.* 2007). In contrast, ERG_e performance is more similar to WSJ results, with a maximum LAS drop of less than two points.[11] For Wikipedia text (WS; previously unseendata for the ERG, just as for the other two), for example, both tagging and dependency accuracies are around ten points higher, an error reduction of more than 50%. From these results, it is evident that the general linguistic knowledge available in ERG parsing makes it far more resilient to variation in domain and text type.

9 ERROR ANALYSIS

The ERG parsers outperform the two data-driven parsers on the WSJ and cross-domain data. Through in-depth error analysis, we seek to identify parser-specific properties that can explain the observed differences. In the following, we look at (a) the accuracy of individual dependency types, (b) dependency accuracy relative to (predicted and gold) dependency length, and (c) the distribution of LAS over different lexical categories.

Among the different dependency types, we observe that the notion of an adjunct is difficult for all three parsers. One of the hardest

[11] It must be noted that, unlike the WSJ test data, some of these cross-domain data sets have been used in ERG development throughout the years, notably VM and CB, and thus the grammar is likely to have particularly good linguistic coverage of this data. Conversely, SC has hardly had a role in grammar engineering so far, and WS is genuinely unseen (for the ERG and Redwoods release used here), i.e. treebankers were first exposed to it once the grammar and parser were frozen.

		Gaps	TA_c	LAS_c	UAS_c
CB	Berkeley	1+0	87.1	78.13	83.14
	B&N	0+0	87.06	77.70	82.96
	ERG_a	0+4	96.3	90.77	92.47
	ERG_e	8+8	95.3	90.02	91.58
SC	Berkeley	1+0	87.2	79.81	85.10
	B&N	0+0	85.9	78.08	83.21
	ERG_a	0+0	96.1	90.84	92.65
	ERG_e	11+7	94.9	89.49	91.26
VM	Berkeley	7+0	84.0	74.40	83.38
	B&N	0+0	83.1	75.28	82.86
	ERG_a	0+40	94.3	90.44	92.27
	ERG_e	11+42	94.4	90.18	91.75
WS	Berkeley	7+0	87.7	80.31	85.11
	B&N	0+0	88.4	80.63	85.24
	ERG_a	0+0	97.5	91.33	92.48
	ERG_e	4+12	96.9	90.64	91.76

Table 5:
Cross-domain coverage gaps (parse failures and token mismatches) and tagging and dependency accuracies

dependency labels across domains is hdn-aj (post-adjunction to a nominal head), the relation employed for relative clauses and prepositional phrases attaching to a nominal head. The most common error for this relation is verbal attachment.

It has been noted that dependency parsers may exhibit systematic performance differences with respect to dependency length (i.e. the distance between a head and its argument; McDonald and Nivre 2007). In our experiments, we find that the parsers perform comparably on longer dependency arcs (upwards of fifteen words), with ERG_a constantly showing the highest accuracy, and Berkeley holding a slight edge over B&N as dependency length increases.

In Figure 3, one can eyeball frequency and accuracy levels per lexical category on WSJ. The cross-domain picture is similar to the in-domain one, but the difference between accuracy for PET and the data-driven parsers on adjectives (aj), adverbs (av), and conjunctions (c) is more pronounced on the out-of-domain data. Determiners (d) and complimentizers (cm) are similar, while conjunctions (c) and various types of prepositions (p and pp) are the most difficult for all three parsers.

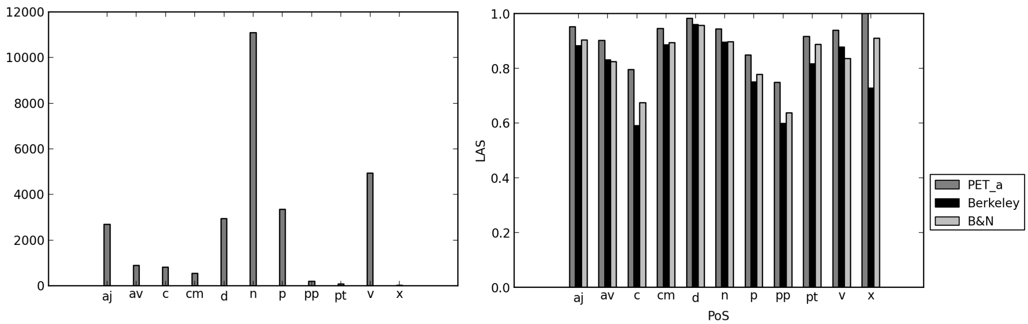


Figure 3: WSJ per-category frequency (left) and dependency accuracies (right) on coarse lexical head categories: adjective, adverb, conjunction, complementizer, determiner, noun, preposition, lexical prepositional phrase, punctuation, verb, and others

It is unsurprising that the DT analysis of coordination is challenging. Schwartz *et al.* (2012) show that choosing conjunctions as heads in coordinate structures is harder to parse for direct dependency parsers (while this analysis also is linguistically more expressive). Our results confirm this effect also for the PCFG parser and (though to a lesser degree) for ERG_a. At the same time, conjunctions are among the lexical categories for which ERG_a most clearly outperforms the other parsers. Berkeley and B&N exhibit LAS error rates of around 35–41% for conjunctions, whereas the ERG_a error rate is below 20%. For many of the coordinate structures parsed correctly by ERG_a but not the other two, we find that attachment to root constitutes the most frequent error type – indicating that clausal coordination is particularly difficult for the data-driven parsers.

The attachment of prepositions constitutes a notorious difficulty in syntactic analysis. Unlike 'standard' PoS tag sets, ERG lexical types provide a more fine-grained analysis of prepositions, for example recognizing a lexicalized PP like *in full*, or making explicit the distinction between semantically contentful vs. vacuous prepositions. In our error analysis, we find that parser performance varies a lot across the various prepositional sub-types. For some prepositions, all parsers perform comparatively well; e.g. p_np_ptcl-of_le, for semantically vacuous *of*, ranks among the twenty most accurate lexical categories across the board. Other types of prepositions are among the categories exhibiting the highest error rates, e.g. p_np_i_le for 'common' prepositions, taking an NP argument and projecting intersective modifier semantics. Even so, Figure 3 shows that the attachment of prepositions (p and pp) is an area where ERG_a excels most markedly. Three frequent prepositional lexical types that show the largest ERG_a advantages are p_np_ptcl-of_le (*history of Linux*), p_np_ptcl_le (*look for peace*), and p_np_i_le (*talk about friends*).

Looking more closely at inputs where the parsers disagree, they largely involve (usages of) prepositions which are lexically selected for by their head. ERG lexical rules, which function as lexical types in DT, encode valency information in the form of an ordered sequence of complements for the given type. For example, v_np-pp_prop_le is a verb that requires two complements: a noun phrase and a prepositional phrase (see example (1)).

We analyze parse errors on prepositional complements for heads of various lexical types, including the most frequent verbs, nouns, and adjectives, illustrated by (1), (2), and (3). Example (1) depicts the analysis of the argument structure of such a verb (*sneak*) with a noun phrase and a prepositional phrase. Both B&N and Berkeley incorrectly define the head of the phrase *into the office* as the noun *therapists*, while ERG_a delivers the parse tree that corresponds to the gold standard. In example (2) ERG_a correctly identifies *growth* as the head of the prepositional phrase *of recent years* while B&N attaches *of* to the cardinal *4* and Berkeley to the conjunction *but* with erroneous dependency labels. In example (3), ERG_a correctly analyzes the prepositional complement, and B&N and Berkeley predict the proper label, but wrongly assign attachment to the noun *work* and verb *sounds*, respectively.

HD-CMP
HD-CMP

(1) … managers *sneak* massage therapists **into** the office …
v_np-pp_prop_le

HD-CMP

(2) … below the 4 % to 5 % *growth* **of** recent years - but …
n_pp_mc-of_le

HD-CMP

(3) … sounds *more* like a shaggy poet describing his work **than** …
aj_pp_i-more_le

In most cases the lexical category of the head explicitly shows the requirement of a prepositional complement; taking advantage of this rule, ERG_a consistently outperforms other parsers in- and cross-domain as depicted in Table 6 which shows the number of total and correct analyses of prepositional complement structures.

Table 6:
Number of total and correct analyses of prepositional complement structures

domain	total	ERG_a	Berkeley	B&N
WSJ	940	**905**	778	799
CB	469	**446**	348	354
SC	602	**553**	471	454
VM	164	**142**	113	119
WS	372	**361**	293	289

Most prepositions in isolation are ambiguous lexical items. However, it appears that lexical information about the argument structure of heads encoded in the grammar allows ERG_a to analyze these prepositions (in context) much more accurately.

10 SANITY: PTB TOKENIZATION AND POS TAGS

Up to this point, we have applied the two data-driven parsers in a setup that one might consider somewhat 'off-road'; although our experiments are on English, they involve unusual tokenization and lexical categories. For example, the ERG treatment of punctuation as 'pseudo-affixes' increases vocabulary size, which PET may be better equipped to handle due to its integrated treatment of morphological variation. In these experiments, we seek to isolate the effects of tokenization conventions and granularity of lexical categories, taking advantage of optional output flexibility in the DT converter of Ivanova *et al.* (2012).[12] Table 7 confirms that tokenization does make a difference. In combination with fine-grained lexical categories still, B&N obtains LAS gains of two to three points, compared to smaller gains (around or below one point) for ERG_e.[13] However, in this setup our

[12] As mapping from ERG derivations into PTB-style tokens and PoS tags is applied when converting to bi-lexical dependencies, we cannot easily include Berkeley in these final experiments.

[13] When converting to PTB-style tokenization, punctuation marks are always attached low in the DT scheme, to the immediately preceding or following to-

two earlier observations still hold true: ERG_e is substantially more accurate within the WSJ domain and far more resilient to domain and genre variation. When we simplify the syntactic analysis task and train and test B&N on coarse-grained PTB PoS tags only, in-domain differences between the two parsers are further reduced (to 0.8 points), but ERG_e still delivers an error reduction of ten percent compared to B&N. The picture in the cross-domain comparison is not qualitatively different, also in this simpler parsing task, with ERG_e maintaining accuracy levels comparable to WSJ, while B&N accuracies degrade markedly.

		Gaps	Lexical Types		PTB PoS Tags	
			LAS_c	UAS_c	LAS_c	UAS_c
WSJ	B&N	0+0	88.78	91.52	91.56	93.63
	ERG_e	13+9	92.38	93.53	92.38	93.53
CB	B&N	0+0	81.56	86.18	84.54	88.53
	ERG_e	8+4	90.77	92.21	90.77	92.21
SC	B&N	0+0	81.69	86.11	85.17	88.85
	ERG_e	11+0	90.13	91.86	90.13	91.86
VM	B&N	0+0	77.00	83.73	82.76	88.11
	ERG_e	10+0	91.55	93.08	91.55	93.08
WS	B&N	0+0	82.09	86.17	84.59	88.41
	ERG_e	4+0	91.61	92.62	91.61	92.62

Table 7:
Coverage and dependency accuracies with PTB tokenization and either detailed or coarse lexical categories

11 FIRST EXTRINSIC EVALUATION: NEGATION SCOPE RESOLUTION

One reason for using a representation format like DT is to make it easy to use parsing results in a downstream application, since parsing is rarely the final goal. In order to test the suitability of DT and also explore the effects that improved parser accuracy may have in such a downstream application, we embed our different parsing setups in an extrinsic evaluation scenario.

Elming *et al.* (2013) try a number of different tasks to explore the effects of different dependency representation formats. They find the

ken, effectively adding a large group of ‘easy’ dependencies. However, results of evaluation excluding punctuation tokens are not qualitatively different.

negation resolution task (Morante and Blanco 2012) to be the most sensitive to changes in dependency format, and so we chose that as our first external task.

11.1 *Negation resolution task*

Negation resolution (NR) is the task of determining negation cues, i.e. morphemes, words or a combination of words that express negation (for example, *un-*, *no*, *by no means*), scopes of negation, i.e. parts of a sentence that are affected by a negation cue, and negated events, i.e. semantically negated eventualities inside scopes in factual sentences. We employ the NR system of Lapponi *et al.* (2012a), one of the best performing systems of the 2012 Computational Semantics (*SEM) Shared Task (Morante and Blanco 2012) which uses a CRF model for scope resolution relying on dependency features. The dataset for the 2012 *SEM shared task comprises negation annotated stories of Conan Doyle: a training set of 3644 sentences, a development set of 787 sentences, and a test set of 1089 sentences. One example from the training set is presented in (4) below. The cue is typeset in small caps, its scope italicized, and the negated event underlined.

(4) *I* therefore spent the day at my club and *did* NOT <u>return</u> *to Baker Street until evening.*

Note that this negation scope is discontinuous, which shows that proximity to a negation cue is not always a good strategy to assign tokens to scopes; here the subject (*I*) at the beginning of the sentence is a part of the clause negated by the cue in the coordinate sentence and should be retrieved.

For the evaluation we use software developed by the 2012 *SEM Shared Task organizers which reports the following metrics (Morante and Blanco 2012):

Cues Cue F_1-measure.

Scopes Scope-level F_1-measure.

Negated F_1-measure over negated events, computed independently from cues and from scopes

Global Global F_1-measure of negation: the three elements of the negation – cue, scope, and negated event – all have to be correctly identified (strict match)

11.2 *Format comparison*

Table 8 presents evaluation of performance of the NR system relying on dependency features from the analyses of the B&N parser with the three dependency formats we tested in Section 3: CoNLL, Stanford Basic, and DT dependencies. As Elming *et al.* (2013) saw, we get quite a range of performance across the three formats, particularly considering Table 1 showed that intrinsic parse accuracy is basically identical.

	CoNLL	Stanford	DT
Scopes	79.57	**81.69**	80.43
Negated	**75.96**	71.15	73.33
Global	**65.89**	63.78	**65.89**

Table 8:
Performance of the NR system with gold cues on the Conan Doyle development set for three dependency formats using the B&N parser

Table 9 reproduces the numbers Elming *et al.* (2013) reported, using dependency formats that varied more than ours do. While these numbers are not directly comparable to our work due to some differences in the data sets for training parsing models, DT is well within their range of variation, and as such, seems a reasonable format for the task.

	Yamada	CoNLL-07	EWT	LTH
Scopes	**81.27**	80.43	78.70	79.57
Negated	76.19	72.90	73.15	**76.24**
Global	**67.94**	63.24	61.60	64.31

Table 9:
Performance of the NR system with gold cues on the Conan Doyle development set for different dependency formats using the Mate parser, reproduced from Elming *et al.*

11.3 *Parser comparison*

To see if the intrinsic parser accuracy differences we saw earlier translate to better negation resolution, we use the PET and B&N parsers to produce DT dependencies for our NR system.

Intrinsic parser evaluation on the 91 manually annotated sentences taken from the story *Wisteria Lodge*, a subset of Conan Doyle development data, is shown in Table 10. Since negation resolution system uses PTB tokenization with PTB PoS tags, we again cannot include Berkeley in this comparison. The Conan Doyle domain is genuinely new for the ERG as it was not available before the release of

Table 10:
Parse failures and token mismatches ('gaps'), and tagging and dependency accuracy on the subset of the Conan Doyle development data

	Gaps	TA_c	LAS_c	UAS_c
B&N	0+0	92.24	83.92	87.92
ERG_a	0+0	**96.36**	**92.54**	**93.84**
ERG_e	0+3	94.21	89.22	90.57

version 1212, used in the present work. Consistent with our expectations, results are similar to the cross-domain evaluation in Table 7.

While B&N has complete coverage on the full Conan Doyle corpus, Table 11 shows that both of our PET variants sometimes fail to produce an analysis, especially the ERG_e variant due to excessive pruning. In addition, PET does not always land on the gold-standard tokenization as the parsing process starts from the raw text. Due to this, we fall back on the B&N parser for the sentences that lack syntactic analysis in the negation resolution experiments with PET; e.g. for the experiments with ERG_a the training set consists of 89.24% PET analyses and 10.76% analyses from B&N.

Table 11:
PET coverage on Conan Doyle and alignment with 'gold' tokenization

	ERG_a			ERG_e		
	Train	Dev	Test	Train	Dev	Test
% Coverage	89.96	91.11	87.42	81.64	83.99	79.98
% Alignment	89.24	91.11	86.23	80.98	83.86	78.88

Tables 12 and 13 show the results of the NR system on the development and test sets, respectively. The results from the original system using the Malt parser and Stanford Basic dependencies are shown for comparison Lapponi *et al.* (2012b). Somewhat surprisingly, the reasonably large differences in parser accuracy seen in Table 7 are not reflected in the task performance. Statistical significance testing using the paired, two-tailed formulation of the sign test shows that none of the performance differences are actually significant.

Table 12:
Performance of the negation resolution system on the development set with gold cues

	ERG_a	ERG_e	B&N	Malt
Scopes	80.00	**80.85**	80.43	80.00
Negated	**75.73**	73.33	73.33	80.55
Global	64.31	63.24	**65.89**	66.41

	ERG_a	ERG_e	B&N	Malt
Cues	91.31	91.31	91.31	91.31
Scopes	73.52	74.83	**75.40**	72.39
Negated	**61.29**	60.95	60.44	61.79
Global	53.73	**55.53**	55.17	54.82

Table 13:
Performance of the negation resolution system on the test set with predicted cues

It is possible that this negation resolution task is not sensitive enough to parser performance to be a useful extrinsic parser evaluation. There is a reasonable body of previous work (Mollá and Hutchinson 2003; Miyao *et al.* 2008; Miwa *et al.* 2010) that has shown that many tasks such as answer extraction, protein-protein interaction (PPI) extraction, and event extraction are relatively insensitive to parser accuracy. It is possible that negation resolution, at least in this particular setup, is another such task.

12 SECOND EXTRINSIC EVALUATION: SEMANTIC DEPENDENCY PARSING

As another downstream application for extrinsic evaluation, we explore the task of Broad-Coverage Semantic Dependency Parsing (SDP; Oepen *et al.* 2014, 2015), which was part of the 2014 and 2015 Semantic Evaluation Exercises (SemEval). We re-train and evaluate the best-performing system from the SDP 2014 open track, called Priberam (Martins and Almeida 2014), which is based on a feature-rich model that takes advantage of the information from the syntactic dependency parser. For this second extrinsic evaluation, we test whether syntactic dependency features provided by the grammar-based system facilitate more accurate semantic parsing than features delivered by the data-driven tools.

12.1 *Broad-coverage semantic dependency parsing*

Oepen *et al.* (2014) define semantic dependency parsing (SDP) as the problem of recovering sentence-internal predicate-argument relationships for all content words. Thus, target representations are semantic in nature (rather than directly representing grammatical structure), and in contrast to syntactic parsing the SDP semantic dependencies are general (directed and acyclic) graphs rather than trees, and need not span input tokens that are analyzed as semantically vacuous.

The SDP 2014 data consists of Sections 0–21 of the WSJ Corpus annotated with three target representations called DM, PAS, and PCEDT (which are all aligned at the sentence and token levels). DM is the result of a two-step reduction of the underspecified logical-form meaning representations produced by the ERG to pure bi-lexical dependencies (Oepen and Lønning 2006; Ivanova *et al.* 2012), as exemplified for our running example in Figure 4. PAS dependencies are predicate–argument structures produced by the Enju system, a statistical HPSG parser obtained by learning from a conversion of the (full) PTB annotations (Miyao and Tsujii 2008). PCEDT dependencies, in turn, are extracted from the tectogrammatical analysis layer of the Prague Czech–English Dependency Treebank (Hajič *et al.* 2012).

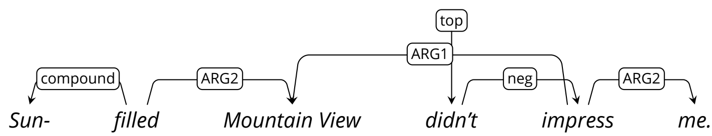


Figure 4: DM bi-lexical semantic dependencies for our running example

The task is organized into two tracks: systems in the *closed* track were trained only on the data distributed by the task organizers while the systems in the *open* track could use additional resources. We are, therefore, only interested in the latter track. In the open track of the SDP 2014 task, participants had been offered syntactic 'companion' files with Stanford dependencies produced by the parser of Bohnet and Nivre (2012). Evaluation measures are labeled precision (LP), labeled recall (LR), labeled F_1 (LF), and labeled exact match (LM) with respect to predicted $\langle$*predicate*, *role*, *argument*$\rangle$ triples.

The Priberam system (Martins and Almeida 2014), which relies on a model with second-order features and decoding with dual decomposition, was ranked first in the SDP 2014 open track, and achieved the second highest score in the closed track. By virtue of syntactic features extracted from the output of a syntactic dependency parser, Priberam attained an improvement of around 1% in LF for all three dependency formats. We have chosen this system for extrinsic evaluation.

12.2 *Parser comparison*

We compare the quality of syntactic features produced by ERG_a, ERG_e and B&N for the semantic parsing with Priberam. Using these three parsers we prepare alternate companion files containing DT bi-lexical dependencies. Of the 1348 sentences in the SDP test set, ERG_a and ERG_e fail to deliver analysis for 11 and 24 sentences, respectively; thus, we 'borrow' the missing analyses from B&N outputs, much like we did in Section 11 above.

Tables 14, 15, and 16 present SDP results for DM, PAS, and PCEDT, respectively. For comparison, we include results of Priberam from the SDP 2014 task with the original companion file generated by task organizers. Compared to the original SDP 2014 results, moving from Stanford to DT dependencies (derived by B&N in both cases) appears to only have a small effect on semantic dependency parsing. Our re-trained version of Priberam with the DT syntactic companion performs marginally below the published SDP 2014 results for

	ERG_a	ERG_e	B&N	SDP 2014
LP	**90.88**	90.77	88.96	90.23
LR	**89.86**	89.67	88.10	88.11
LF	**90.37**	90.22	88.53	89.16
LM	**32.42**	32.64	29.75	26.85

Table 14:
SDP open track results on DM

	ERG_a	ERG_e	B&N	SDP 2014
LP	92.04	**92.19**	91.91	92.56
LR	89.67	**89.89**	89.63	90.97
LF	90.84	**91.03**	90.75	91.76
LM	31.38	30.93	**32.86**	37.83

Table 15:
SDP open track results on PAS

	ERG_a	ERG_e	B&N	SDP 2014
LP	79.62	**79.94**	79.42	80.14
LR	75.67	**75.82**	75.45	75.79
LF	77.59	**77.82**	77.38	77.90
LM	11.05	**11.20**	10.98	10.68

Table 16:
SDP open track results on PCEDT

DM and PCEDT, whereas for PAS there is a more pronounced drop in semantic dependency LF when replacing Stanford with DT dependencies. Different syntactic accuracy levels of our three DT parsers, on the other hand, actually do project into downstream differences in semantic dependency quality: For all three target representations, the ERG parsers yield higher semantic dependency LF than B&N. The differences are comparatively small for the PAS and PCEDT targets, but for DM there is a large benefit in deriving the (more accurate) DT syntactic companion from ERG_a rather than from B&N. Seeing as DM and DT both originate from DeepBank, while PAS as well as Stanford dependencies originate from the PTB, our results suggest that it is beneficial for the semantic dependency parsing task to rely on 'correlated' syntactic dependency features: The overall best-performing SDP pipeline for the DM target representation uses DT dependencies (from ERG_a), but the best PAS results are obtained with Stanford syntactic dependencies (from B&N).

In conclusion, the results of this second extrinsic evaluation suggest that semantic dependency parsing is more sensitive to syntactic parser performance than negation resolution, especially when taking into account that the maximum in-domain difference between ERG_e and B&N observed in Table 7 is 0.82% in LAS_c when using PTB tokenization and PTB PoS tags (as is also the case in the SDP 2014 task).

13 DISCUSSION AND CONCLUSION

Our experiments have sought to contrast state-of-the-art representatives from three parsing paradigms on the task of producing bi-lexical syntactic dependencies for English. For the HPSG-derived DT scheme, we find that hybrid, grammar-driven parsing yields superior accuracy, both in- and in particular cross-domain, at processing times comparable to the currently best direct dependency parser; the grammar-driven parser in our experiments, however, fails to parse a small percentage of inputs in naturally occurring text. These results corroborate the Dutch findings of Plank and van Noord (2010) for English, where more training data is available, and in comparison to more advanced data-driven parsers. Extrinsic evaluation on semantic dependency parsing correlates with the results of the in-domain intrinsic evaluation. However, we do not find that this superior accuracy is reflected in superior accuracy in the negation resolution task. In most of this work, we have focussed exclusively on parser inputs represented in the DeepBank and Redwoods treebanks, ignoring 15 percent of the original running text, for which the ERG and PET do not make available a gold-standard analysis. While a parser with partial coverage can be useful in some contexts, obviously the data-driven parsers must be credited for providing a syntactic analysis of (almost) all inputs. However, the ERG coverage gap can be straightforwardly addressed by falling back to another parser when necessary, as we did in our extrinsic evaluations. Such a system combination should yield better tagging and dependency accuracies than the data-driven parsers by themselves, especially so in an open-domain setup. A secondary finding from our experiments is that PCFG parsing with Berkeley and conversion to DT dependencies yields equivalent or mildly more accurate analyses, at much greater efficiency. In future work, it would be interesting to include in this comparison other PCFG parsers and linear-time, transition-based dependency parsers, but a tentative generalization over our findings to date is that linguistically richer representations enable more accurate parsing. It would also be informative to try a wider variety of downstream tasks to see which are sensitive to parser accuracy, as opposed to dependency representation.

ACKNOWLEDGMENTS

We are grateful to our colleagues Emily M. Bender, Bernd Bohnet, Francis Bond, Rui Wang, Yi Zhang, and André Martins for many helpful discussions, suggestions, and assistance in running third-party tools, as well as to our three anonymous reviewers for insightful comments. This work was in part funded by the Norwegian Research Council through its WeSearch project. Large-scale experimentation is made possible through access to the ABEL high-performance computing facilities at the University of Oslo, and we are grateful to the Scientific Computing staff at UiO, as well as to the Norwegian Metacenter for Computational Science, and the Norwegian tax payer.

REFERENCES

Peter ADOLPHS, Stephan OEPEN, Ulrich CALLMEIER, Berthold CRYSMANN, Dan FLICKINGER, and Bernd KIEFER (2008), Some Fine Points of Hybrid Natural Language Parsing, in *Proceedings of the 6th International Conference on Language Resources and Evaluation*, Marrakech, Morocco.

Ezra BLACK, Steve ABNEY, Dan FLICKINGER, Claudia GDANIEC, Ralph GRISHMAN, Phil HARRISON, Don HINDLE, Robert INGRIA, Fred JELINEK, Judith KLAVANS, Mark LIBERMAN, Mitch MARCUS, S. ROUKOS, Beatrice SANTORINI, and Tomek STRZALKOWSKI (1991), A Procedure for Quantitatively Comparing the Syntactic Coverage of English Grammars, in *Proceedings of the Workshop on Speech and Natural Language*, pp. 306–311, Pacific Grove, USA.

Bernd BOHNET and Joakim NIVRE (2012), A Transition-Based System for Joint Part-of-Speech Tagging and Labeled Non-Projective Dependency Parsing, in *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Conference on Natural Language Learning,* pp. 1455–1465, Jeju Island, Korea.

Johan BOS, Edward BRISCOE, Aoife CAHILL, John CARROLL, Stephen CLARK, Ann COPESTAKE, Dan FLICKINGER, Josef VAN GENABITH, Julia HOCKENMAIER, Aravind JOSHI, Ronald KAPLAN, Tracy Holloway KING, Sandra KUEBLER, Dekang LIN, Jan Tore LØNNING, Christopher MANNING, Yusuke MIYAO, Joakim NIVRE, Stephan OEPEN, Kenji SAGAE, Nianwen XUE, and Yi ZHANG, editors (2008), *Workshop on Cross-Framework and Cross-Domain Parser Evaluation*, Manchester, UK.

Ted BRISCOE and John CARROLL (2006), Evaluating the Accuracy of an Unlexicalised Statistical Parser on the PARC DepBank, in *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Meeting of the Association for Computational Linguistics*, pp. 41–48, Sydney, Australia.

Ulrich CALLMEIER (2002), Preprocessing and Encoding Techniques in PET, in Stephan OEPEN, Daniel FLICKINGER, J. TSUJII, and Hans USZKOREIT, editors, *Collaborative Language Engineering. A Case Study in Efficient Grammar-Based Processing*, pp. 127–140, CSLI Publications, Stanford, CA.

David CARTER (1997), The TreeBanker. A Tool for Supervised Training of Parsed Corpora, in *Proceedings of the Workshop on Computational Environments for Grammar Development and Linguistic Engineering*, pp. 9–15, Madrid, Spain.

Daniel CER, Marie-Catherine DE MARNEFFE, Dan JURAFSKY, and Chris MANNING (2010), Parsing to Stanford Dependencies. Trade-Offs between Speed and Accuracy, in *Proceedings of the 7th International Conference on Language Resources and Evaluation*, pp. 1628–1632, Valletta, Malta.

Stephen CLARK and James R. CURRAN (2007), Formalism-Independent Parser Evaluation with CCG and DepBank, in *Proceedings of the 45th Meeting of the Association for Computational Linguistics*, pp. 248–255, Prague, Czech Republic.

Marie-Catherine DE MARNEFFE and Christopher D. MANNING (2008), The Stanford Typed Dependencies Representation, in *Proceedings of the COLING Workshop on Cross-Framework and Cross-Domain Parser Evaluation*, pp. 1–8, Manchester, UK.

Rebecca DRIDAN (2013), Ubertagging. Joint Segmentation and Supertagging for English, in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1–10, Seattle, WA, USA.

Jacob ELMING, Anders JOHANNSEN, Sigrid KLERKE, Emanuele LAPPONI, Hector MARTINEZ, and Anders SØGAARD (2013), Down-Stream Effects of Tree-to-Dependency Conversions, in *Proceedings of Human Language Technologies: The 2013 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 617–626, Atlanta, GA, USA.

Dan FLICKINGER (2000), On Building a More Efficient Grammar by Exploiting Types, *Natural Language Engineering*, 6 (1):15–28.

Dan FLICKINGER, Yi ZHANG, and Valia KORDONI (2012), DeepBank. A Dynamically Annotated Treebank of the Wall Street Journal, in *Proceedings of the 11th International Workshop on Treebanks and Linguistic Theories*, pp. 85–96, Edições Colibri, Lisbon, Portugal.

Jennifer FOSTER, Ozlem CETINOGLU, Joachim WAGNER, Joseph LE ROUX, Joakim NIVRE, Deirdre HOGAN, and Josef VAN GENABITH (2011), From News to Comment. Resources and Benchmarks for Parsing the Language of Web 2.0, in *Proceedings of the 2011 International Joint Conference on Natural Language Processing*, pp. 893–901, Chiang Mai, Thailand.

Timothy A. D. FOWLER and Gerald PENN (2010), Accurate Context-Free Parsing with Combinatory Categorial Grammar, in *Proceedings of the 48th Meeting of the Association for Computational Linguistics*, pp. 335–344, Uppsala, Sweden.

W. Nelson FRANCIS and Henry KUČERA (1982), *Frequency Analysis of English Usage. Lexicon and Grammar*, Houghton Mifflin, New York, USA.

Daniel GILDEA (2001), Corpus Variation and Parser Performance, in *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing*, pp. 167–202, Pittsburgh, USA.

Jan HAJIČ, Eva HAJIČOVÁ, Jarmila PANEVOVÁ, Petr SGALL, Ondřej BOJAR, Silvie CINKOVÁ, Eva FUČÍKOVÁ, Marie MIKULOVÁ, Petr PAJAS, Jan POPELKA, Jiří SEMECKÝ, Jana ŠINDLEROVÁ, Jan ŠTĚPÁNEK, Josef TOMAN, Zdeňka UREŠOVÁ, and Zdeněk ŽABOKRTSKÝ (2012), Announcing Prague Czech-English Dependency Treebank 2.0, in *Proceedings of the 8th International*

Conference on Language Resources and Evaluation, pp. 3153–3160, Istanbul, Turkey.

Julia HOCKENMAIER and Mark STEEDMAN (2007), CCGbank. A Corpus of CCG Derivations and Dependency Structures Extracted from the Penn Treebank, *Computational Linguistics*, 33:355–396.

Angelina IVANOVA, Stephan OEPEN, and Lilja ØVRELID (2013), Survey on Parsing Three Dependency Representations for English, in *Proceedings of the 51th Meeting of the Association for Computational Linguistics*, pp. 31–37, Sofia, Bulgaria.

Angelina IVANOVA, Stephan OEPEN, Lilja ØVRELID, and Dan FLICKINGER (2012), Who Did What to Whom? A Contrastive Study of Syntacto-Semantic Dependencies, in *Proceedings of the Sixth Linguistic Annotation Workshop*, pp. 2–11, Jeju, Republic of Korea.

Tracy Holloway KING, Richard CROUCH, Stefan RIEZLER, Mary DALRYMPLE, and Ronald M. KAPLAN (2003), The PARC 700 Dependency Bank, in *Proceedings of the 4th International Workshop on Linguistically Interpreted Corpora*, pp. 1–8, Budapest, Hungary.

Emanuele LAPPONI, Jonathon READ, and Lilja ØVRELID (2012a), Representing and Resolving Negation for Sentiment Analysis, in *Proceedings of the 2012 ICDM Workshop on Sentiment Elicitation from Natural Text for Information Retrieval and Extraction*, Brussels, Belgium.

Emanuele LAPPONI, Erik VELLDAL, Lilja ØVRELID, and Jonathon READ (2012b), UiO2: Sequence-Labeling Negation Using Dependency Features, in *Proceedings of the 1st Joint Conference on Lexical and Computational Semantics*, pp. 319–327, Montréal, Canada.

Mitchell MARCUS, Beatrice SANTORINI, and Mary Ann MARCINKIEWICZ (1993), Building a Large Annotated Corpora of English. The Penn Treebank, *Computational Linguistics*, 19:313–330.

T. André F. MARTINS and C. Mariana S. ALMEIDA (2014), Priberam: A Turbo Semantic Parser with Second Order Features, in *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pp. 471–476, Association for Computational Linguistics, Dublin, Ireland.

Ryan T. MCDONALD and Joakim NIVRE (2007), Characterizing the Errors of Data-Driven Dependency Parsing Models, in *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Conference on Natural Language Learning*, pp. 122–131, Prague, Czech Republic.

Makoto MIWA, Sampo PYYSALO, Tadayoshi HARA, and Jun'ichi TSUJII (2010), Evaluating Dependency Representations for Event Extraction, in *Proceedings of the 23rd International Conference on Computational Linguistics*, pp. 779–787.

Yusuke MIYAO, Rune SÆTRE, Kenji SAGAE, Takuya MATSUZAKI, and Jun'ichi TSUJII (2008), Task-Oriented Evaluation of Syntactic Parsers and Their Representations, in *Proceedings of the 46th Meeting of the Association for Computational Linguistics*, pp. 46–54, Columbus, OH, USA.

Yusuke MIYAO, Kenji SAGAE, and Jun'ichi TSUJII (2007), Towards Framework-Independent Evaluation of Deep Linguistic Parsers, in *Proceedings of the 2007 Workshop on Grammar Engineering across Frameworks*, pp. 238–258, Palo Alto, California.

Yusuke MIYAO and Jun'ichi TSUJII (2008), Feature Forest Models for Probabilistic HPSG Parsing, *Computational Linguistics*, 34(1):35–80.

Diego MOLLÁ and Ben HUTCHINSON (2003), Intrinsic Versus Extrinsic Evaluations of Parsing Systems, in *Proceedings of the EACL 2003 Workshop on Evaluation Initiatives in Natural Language Processing: Are Evaluation Methods, Metrics and Resources Reusable?*, pp. 43–50, Budapest, Hungary.

Roser MORANTE and Eduardo BLANCO (2012), *SEM 2012 Shared Task. Resolving the Scope and Focus of Negation, in *Proceedings of the 1st Joint Conference on Lexical and Computational Semantics*, pp. 265–274, Montréal, Canada.

Joakim NIVRE, Johan HALL, Sandra KÜBLER, Ryan MCDONALD, Jens NILSSON, Sebastian RIEDEL, and Deniz YURET (2007), The CoNLL 2007 Shared Task on Dependency Parsing, in *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Conference on Natural Language Learning*, pp. 915–932, Prague, Czech Republic.

Stephan OEPEN and John CARROLL (2000), Ambiguity Packing in Constraint-Based Parsing. Practical Results, in *Proceedings of the 1st Meeting of the North American Chapter of the Association for Computational Linguistics*, pp. 162–169, Seattle, WA, USA.

Stephan OEPEN, Daniel FLICKINGER, Kristina TOUTANOVA, and Christopher D. MANNING (2004), LinGO Redwoods. A Rich and Dynamic Treebank for HPSG, *Research on Language and Computation*, 2(4):575–596.

Stephan OEPEN, Marco KUHLMANN, Yusuke MIYAO, Daniel ZEMAN, Silvie CINKOVA, Dan FLICKINGER, Jan HAJIC, and Zdenka URESOVA (2015), SemEval 2015 Task 18: Broad-Coverage Semantic Dependency Parsing, in *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pp. 915–926, Denver, CO, USA.

Stephan OEPEN, Marco KUHLMANN, Yusuke MIYAO, Daniel ZEMAN, Dan FLICKINGER, Jan HAJIČ, Angelina IVANOVA, and Yi ZHANG (2014), SemEval 2014 Task 8. Broad-Coverage Semantic Dependency Parsing, in *Proceedings of the 8th International Workshop on Semantic Evaluation*, Dublin, Ireland.

Stephan OEPEN and Jan Tore LØNNING (2006), Discriminant-Based MRS Banking, in *Proceedings of the 5th International Conference on Language Resources and Evaluation*, pp. 1250–1255, Genoa, Italy.

Slav PETROV, Leon BARRETT, Romain THIBAUX, and Dan KLEIN (2006), Learning Accurate, Compact, and Interpretable Tree Annotation, in *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Meeting of the Association for Computational Linguistics*, pp. 433–440, Sydney, Australia.

Barbara PLANK and Gertjan VAN NOORD (2010), Grammar-Driven versus Data-Driven. Which Parsing System is more Affected by Domain Shifts?, in *Proceedings of the 2010 Workshop on NLP and Linguistics: Finding the Common Ground*, pp. 25–33, Association for Computational Linguistics, Uppsala, Sweden.

Carl POLLARD and Ivan A. SAG (1994), *Head-Driven Phrase Structure Grammar*, Studies in Contemporary Linguistics, The University of Chicago Press, Chicago, USA.

Roy SCHWARTZ, Omri ABEND, and Ari RAPPOPORT (2012), Learnability-Based Syntactic Annotation Design, in *Proceedings of the 24th International Conference on Computational Linguistics*, Mumbai, India.

Wolfgang WAHLSTER, editor (2000), *Verbmobil. Foundations of Speech-to-Speech Translation*, Springer, Berlin, Germany.

Gisle YTRESTØL, Stephan OEPEN, and Dan FLICKINGER (2009), Extracting and Annotating Wikipedia Sub-Domains, in *Proceedings of the 7th International Workshop on Treebanks and Linguistic Theories*, pp. 185–197, Groningen, The Netherlands.

Yi ZHANG and Hans-Ulrich KRIEGER (2011), Large-Scale Corpus-Driven PCFG Approximation of an HPSG, in *Proceedings of the 12th International Conference on Parsing Technologies*, pp. 198–208, Dublin, Ireland.

Yi ZHANG, Stephan OEPEN, and John CARROLL (2007), Efficiency in Unification-Based N-Best Parsing, in *Proceedings of the 10th International Conference on Parsing Technologies*, pp. 48–59, Prague, Czech Republic.

Yi ZHANG and Rui WANG (2009), Cross-Domain Dependency Parsing Using a Deep Linguistic Grammar, in *Proceedings of the 47th Meeting of the Association for Computational Linguistics*, pp. 378–386, Suntec, Singapore.