A synchronous context free grammar for time normalization

Steven Bethard
University of Alabama at Birmingham
Birmingham, Alabama, USA
bethard@cis.uab.edu

Abstract

We present an approach to time normalization (e.g. *the day before yesterday*⇒2013-04-12) based on a synchronous context free grammar. Synchronous rules map the source language to formally defined operators for manipulating times (FINDENCLOSED, STARTATENDOF, etc.). Time expressions are then parsed using an extended CYK+ algorithm, and converted to a normalized form by applying the operators recursively. For evaluation, a small set of synchronous rules for English time expressions were developed. Our model outperforms HeidelTime, the best time normalization system in TempEval 2013, on four different time normalization corpora.

1 Introduction

Time normalization is the task of converting a natural language expression of time into a formal representation of a time on a timeline. For example, the expression *the day before yesterday* would be normalized to the formal representation 2013-04-12 (assuming that today is 2013-04-14) in the ISO-TimeML representation language (Pustejovsky et al., 2010). Time normalization is a crucial part of almost any information extraction task that needs to place entities or events along a timeline. And research into methods for time normalization has been growing since the ACE[1] and TempEval (Verhagen et al., 2010; UzZaman et al., 2013) challenges began to include time normalization as a shared task.

[1]http://www.itl.nist.gov/iad/mig/tests/ace/

Most prior work on time normalization has taken a rule-based, string-to-string translation approach. That is, each word in a time expression is looked up in a normalization lexicon, and then rules map this sequence of lexical entries directly to the normalized form. HeidelTime (Strötgen and Gertz, 2012), which had the highest performance in TempEval 2010 and 2013, and TIMEN (Llorens et al., 2012), which reported slightly higher performance in its own experiments, both follow this approach. A drawback of this approach though is that there is no nesting of rules: for example, in HeidelTime the rules for *yesterday* and *the day before yesterday* are completely separate, despite the compositional nature of the latter.

A notable exception to the string-to-string approach is the work of (Angeli et al., 2012). They define a target grammar of typed pre-terminals, such as YESTERDAY (a SEQUENCE) or DAY (a DURATION), and compositional operations, such as SHIFTLEFT (a (RANGE, DURATION) → RANGE). They apply an expectation-maximization approach to learn how words align to elements of the target grammar, and achieve performance close to that of the rule-based systems. However, their grammar does not allow for non-binary or partially lexicalized rules (e.g. SEQUENCE → DURATION *before* SEQUENCE would be impossible), and some of their primitive elements could naturally be expressed using other primitives (e.g. YESTERDAY as SHIFTLEFT(TODAY, 1 DAY)).

We present a synchronous grammar for time normalization that addresses these shortcomings. We first define a grammar of formal operations over temporal elements. We then develop synchronous rules that map time expression words to temporal opera-

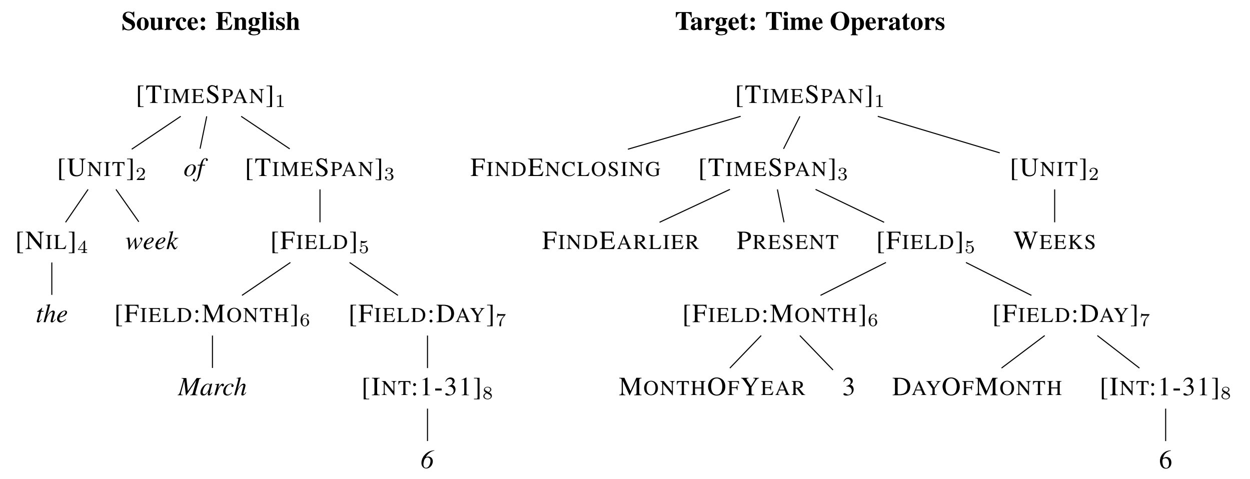

Figure 1: The synchronous parse from the source language *the week of March 6* to the target formal time representation FINDENCLOSING(FINDEARLIER(PRESENT, MONTHOFYEAR→3, DAYOFMONTH→6), WEEKS). Subscripts on non-terminals indicate the alignment between the source and target parses.

tors, and perform normalization by parsing with an extended CYK+ parsing algorithm. We evaluate this approach to time normalization on the TimeBank, AQUAINT, Timen and TempEval 2013 corpora.

2 Synchronous grammars

Our time grammar is based on the synchronous context free grammar formalism. Synchronous grammars allow two trees, one in the source language and one in the target language, to be constructed simultaneously. A synchronous context free grammar has rules of the form $X \rightarrow (\mathbf{S}, \mathbf{T}, A)$, where X is a non-terminal, $\mathbf{S}$ is the sequence of terminals and non-terminals that X expands to in the source language, $\mathbf{T}$ is the sequence of terminals and non-terminals that X expands to in the target language, and A is the alignment between the non-terminals of $\mathbf{S}$ and $\mathbf{T}$ (which must be the same).

For time normalization, the source side is the natural language text, and the target side is a formal grammar of temporal operators. Figure 1 shows a synchronous parse of *the week of March 6* [2]. The left side is the source side (an English expression), the right side is the target side (a temporal operator expression), and the alignment is shown via subscripts.

[2]Figure 1 corresponds to an interpretation along the lines of *the week of the last March 6*. The full grammar developed in this article would also produce an interpretation corresponding to *the week of the next March 6*, since the phrase is ambiguous.

3 Target time grammar

The right side of Figure 1 shows an example of our target formal representation: FINDENCLOSING(FINDEARLIER(PRESENT, MONTHOFYEAR→3, DAYOFMONTH→6), WEEKS). Each terminal in the parse is either a numeric value or an operator like FINDENCLOSING, WEEKS or MONTHOFYEAR. Each non-terminal combines terminals or non-terminals to create a [TIMESPAN], [PERIOD], [FIELD], [UNIT] or [INT]. The list of rules allowed by our target grammar (the right-hand side of our synchronous grammar) is given in Table 1.

Each of the target operators defines a procedure for creating a temporal object from others. For example, FINDENCLOSING takes a [TIMESPAN] and a [UNIT] and expands the start and end of the time span to fill a period of one unit. This could be used, for example, to define *today* as FINDENCLOSING(PRESENT, DAYS), where the PRESENT, which is instantaneous, is expanded out to the enclosing day. Note that we define things like *today* and *yesterday* in terms of primitive operations, rather than making them primitives themselves as in (Angeli et al., 2012).

The left side of Figure 1 shows the synchronous parse of the source language. Note that each of the non-terminals is aligned (shown as a subscript) with a non-terminal in the target parse[3], while terminals are not aligned and may freely appear or disappear

[3]We actually allow a slightly asynchronous grammar, where a non-terminal may be used 0 or more times on the target side.

[Int]	→	integer
[Unit]	→	unit
[Field]	→	field [Int]
[Field]	→	[Field]*
[Period]	→	Simple [Int] [Unit]
[Period]	→	Fractional [Int] [Int] [Unit]
[Period]	→	Unspecified [Unit]
[Period]	→	WithModifier [Period] modifier
[TimeSpan]	→	Past
[TimeSpan]	→	Present
[TimeSpan]	→	Future
[TimeSpan]	→	FindEarlier [TimeSpan] [Field]
[TimeSpan]	→	FindLater [TimeSpan] [Field]
[TimeSpan]	→	FindEnclosing [TimeSpan] [Unit]
[TimeSpan]	→	FindEnclosed [TimeSpan] [Field]
[TimeSpan]	→	StartAtEndOf [TimeSpan] [Period]
[TimeSpan]	→	EndAtStartOf [TimeSpan] [Period]
[TimeSpan]	→	MoveEarlier [TimeSpan] [Period]
[TimeSpan]	→	MoveLater [TimeSpan] [Period]
[TimeSpan]	→	WithModifier [TimeSpan] modifier

Table 1: Rules allowed by the target time grammar. A "unit" is any `java.time.temporal.TemporalUnit`, e.g. Seconds, Weeks or Decades. A "field" is any `java.time.temporal.TemporalField`, e.g. HourOfAmPm, DayOfMonth or Century. A "modifier" is any of the TIMEX3 "mod" values defined in TimeML.

from the source to the target. Each non-terminal thus corresponds to a synchronous grammar rule that describes how a source expression should be translated into the target time grammar. For example the root nodes correspond to an application of the following full synchronous rule:

[TimeSpan] →
source: [Unit] *of* [TimeSpan]
target: FindEnclosing [TimeSpan] [Unit]

4 Parsing algorithm

Parsing with a synchronous context free grammar is much the same as parsing with just the source side of the grammar. Only a small amount of bookkeeping is necessary to allow the generation of the target parse once the source parse is complete. We can therefore apply standard parsing algorithms to this task.

However, we have some additional grammar requirements. As shown in Figure 1, we allow rules that expand into more than two terminals or non-terminals, the mixing of terminals and non-terminals in a production, a special [Nil] non-terminal for the ignoring of words, and a special [Int] non-terminal that can match ranges of integers and does not require all possible integers to be manually listed in the grammar. This means that we can't directly use CYK parsing or even CYK+ parsing (Chappelier and Rajman, 1998), which allows rules that expand into more than two terminals or non-terminals, but does not meet our other requirements.

Algorithm 1 shows our extended version of CYK+ parsing. As with standard CYK+ parsing, two charts are filled, one for rules that have been completed (C) and one for rules that have been only partially advanced (P). All parses covering 1 terminal are completed first, then these are used to complete parses covering 2 terminals, etc. until all parses covering all terminals are complete.

Our extensions to the standard CYK+ parsing are as follows. To handle integers, we modify the initialization to generate new rules on the fly for any numeric terminals that fit the range of an [Int:x-y] non-terminal in the grammar (starts at line 5). To allow mixing of terminals and non-terminals, we extend the initialization step to also produce partial parses (line 17), and extend the parse advancement step to allow advancing rules with terminals (starting at line 23). Finally, to handle [Nil] rules, which consume tokens but are not included in the final parse, we add a step where rules are allowed to advance, unchanged, past a [Nil] rule (starting at line 35).

5 Parsing example

As an example, consider parsing *the week of March 6* with the following source side grammar:

[Nil] → *the*
[Unit] → *week*
[Month] → *March*
[Day] → [Int:1-31]
[Field] → [Month][Day]
[TimeSpan] → [Field]
[TimeSpan] → [Unit] *of* [TimeSpan]

First the algorithm handles the numeric special case, completing an [Int] parse for the token *6* at index 4:

$C_{(1,4)}$ ∪= [Int:1-31] → *6*

Then it completes parses based on just the terminals:

$C_{(1,0)}$ ∪= [Nil] → *the*
$C_{(1,1)}$ ∪= [Unit] → *week*
$C_{(1,3)}$ ∪= [Month] → *March*

Next, the algorithm starts working on parses that span 1 token. It can start two partial parses, using the [Unit] at $C_{(1,1)}$, and using the [Month] at $C_{(1,3)}$:

$P_{(1,1)}$ ∪= [TimeSpan] → [Unit] • *of* [TimeSpan]
$P_{(1,3)}$ ∪= [Field] → [Month] • [Day]

Algorithm 1 CYK+ parsing, extended for partially lexicalized rules, [Nil] rules and numbers

```
Require: $G$ a set of rules, $w$ a sequence of tokens
 1: function PARSE($G, w$)
 2:   $C \leftarrow$ a new $|w|+1$ by $|w|$ matrix
 3:   $P \leftarrow$ a new $|w|+1$ by $|w|$ matrix
 4:   // Generate rules on the fly for numeric tokens
 5:   for $i \leftarrow 0 \dots (|w|-1)$ do
 6:     if ISNUMBER($w_i$) then
 7:       for all [INT:$x$-$y$] $\in$ non-terminals of $G$ do
 8:         if $x \leq$ TONUMBER($w_i$) $\leq y$ then
 9:           $C_{(1,i)} \cup=$ [INT:$x$-$y$] $\rightarrow w_i$
10:   // Start any rules that begin with terminals
11:   for $i \leftarrow 0 \dots (|w|-1)$ do
12:     for all X $\rightarrow \alpha\beta \in G$ do
13:       if $\exists j \mid \alpha = w_{i:j} \wedge \neg$ISTERMINAL($\beta_0$) then
14:         if $\beta = \epsilon$ then
15:           $C_{(|w_{i:j}|,i)} \cup=$ X $\rightarrow w_{i:j}\alpha$
16:         else
17:           $P_{(|w_{i:j}|,i)} \cup= (|w_{i:j}|, \text{X} \rightarrow w_{i:j}\alpha)$
18:   for $n \leftarrow 1 \dots |w|; i \leftarrow 0 \dots (|w|-n)$ do
19:     // Find all parses of size $n$ starting at $i$
20:     for $m \leftarrow 1 \dots n$ do
21:       for all $(p, \text{X} \rightarrow \alpha) \in P_{(m,i)}$ do
22:         // Advance partial parses using terminals
23:         if $w_{i+m:i+n} = \alpha_{p:p+n-m}$ then
24:           if $\alpha_{p+n-m:|\alpha|} = \epsilon$ then
25:             $C_{(n,i)} \cup=$ X $\rightarrow \alpha$
26:           else
27:             $P_{(n,i)} \cup= (p+n-m, \text{X} \rightarrow \alpha)$
28:         // Advance partial parses using completes
29:         for all $\alpha_p \rightarrow \beta \in C_{(n-m,i+m)}$ do
30:           if $|\alpha| = p+1$ then
31:             $C_{(n,i)} \cup=$ X $\rightarrow \alpha$
32:           else
33:             $P_{(n,i)} \cup= (p+1, \text{X} \rightarrow \alpha)$
34:       // Advance complete parses past [Nil] parses
35:       for all X $\rightarrow \alpha \in C_{(m,i)}$ do
36:         for all Y $\rightarrow \beta \in C_{(n-m,i+m)}$ do
37:           if X $\neq$ Nil $\wedge$ Y $=$ Nil then
38:             $C_{(n,i)} \cup=$ X $\rightarrow \alpha$
39:           else if X $=$ Nil $\wedge$ Y $\neq$ Nil then
40:             $C_{(n,i)} \cup=$ Y $\rightarrow \beta$
41:     // Start any rules that begin with a complete parse
42:     for all X $\rightarrow \alpha \in C_{(n,i)}$ do
43:       for all Y $\rightarrow$ X$\alpha \in C_{(n,i)}$ do
44:         if $\alpha = \epsilon$ then
45:           $C_{(n,i)} \cup=$ Y $\rightarrow$ X$\alpha$
46:         else
47:           $P_{(n,i)} \cup= (1, \text{Y} \rightarrow \text{X}\alpha)$
48:   return $C_{(|w|,0)}$
```

(The • is the visual equivalent of the first element in the partial parse tuples of Algorithm 1, which marks parsing progress.) And given the [INT:1-31] at $C_{(1,4)}$ the algorithm can make a complete size 1 parse:

$C_{(1,4)} \cup=$ [DAY] $\rightarrow$ [INT:1-31]

The algorithm then moves on to create parses that span 2 tokens. The special handling of [NIL] allows the [UNIT] at $C_{(1,1)}$ to absorb the [NIL] at $C_{(1,0)}$:

$C_{(2,0)} \cup=$ [UNIT] $\rightarrow$ *week*

This [UNIT] then allows the start of a partial parse:

$P_{(2,0)} \cup=$ [TIMESPAN] $\rightarrow$ [UNIT] • *of* [TIMESPAN]

The partial parse at $P_{(1,1)}$ can be advanced using *of* at position 2, creating another 2 token partial parse:

$P_{(2,1)} \cup=$ [TIMESPAN] $\rightarrow$ [UNIT] *of* • [TIMESPAN])

The partial parse at $P_{(1,3)}$ can be advanced using the [DAY] at $C_{(1,4)}$, completing the 2 token parse:

$C_{(2,3)} \cup=$ [FIELD] $\rightarrow$ [MONTH][DAY]

This [FIELD] allows completion of a 2 token parse:

$C_{(2,3)} \cup=$ [TIMESPAN] $\rightarrow$ [FIELD]

The algorithm then moves on to 3 token parses. Only one is possible: the partial parse at $P_{(2,0)}$ can be advanced using the *of* at position 2, yielding:

$P_{(3,0)} \cup=$ [TIMESPAN] $\rightarrow$ [UNIT] *of* • [TIMESPAN]

The algorithm moves on to 4 token parses, finding that the partial parse at $P_{(2,1)}$ can be advanced using the [TIMESPAN] at $C_{(2,3)}$, completing the parse:

$C_{(4,1)} \cup=$ [TIMESPAN] $\rightarrow$ [UNIT] *of* [TIMESPAN]

Finally, the algorithm moves on to 5 token parses, where (1) the special handling of [NIL] allows the partial parse at $C_{(4,1)}$ to consume the [NIL] at $C_{(1,0)}$ and (2) the partial parse at $P_{(3,0)}$ can be advanced using the [TIMESPAN] at $C_{(2,3)}$. Both of these yield:

$C_{(5,0)} \cup=$ [TIMESPAN] $\rightarrow$ [UNIT] *of* [TIMESPAN]

The complete parses in $C_{(5,0)}$ are then deterministically translated into target side parses using the alignments in the rules of the synchronous grammar.

6 Evaluation

Using our synchronous grammar formalism for time normalization, we manually developed a grammar for English time expressions. Following the lead of TIMEN and HeidelTime, we developed our grammar by inspecting examples from the AQUAINT[4] and

[4] http://www.ldc.upenn.edu/Catalog/catalogEntry.jsp?catalogId=LDC2002T31

	N	TIMEN	HeidelTime	SCFG
AQUAINT	652	69.5	74.7	**76.5**
TimeBank	1426	67.7	80.9	**84.9**
Timen	214	**67.8**	49.1	56.5
TempEval2013	158	74.1	78.5	**81.6**

Table 2: Performance of TIMEN, HeidelTime and our synchronous context free grammar (SCFG) on each evaluation corpus. (N is the number of time expressions.)

TimeBank (Pustejovsky et al., 2003) corpora. The resulting grammar has 354 rules, 192 of which are only lexical, e.g., [UNIT] → (*seconds*, SECONDS).

Our grammar produces multiple parses when the input is ambiguous. For example, the expression *Monday* could mean either the previous Monday or the following Monday, and the expression *the day* could refer either to a period of one day, or to a specific day in time, e.g. 2013-04-14. For such expressions, our grammar produces both parses. To choose between the two, we employ a very simple set of heuristics: (1) prefer [TIMESPAN] to [PERIOD], (2) prefer an earlier [TIMESPAN] to a later one and (3) prefer a [TIMESPAN] with QUARTERS granularity if the anchor time is also in QUARTERS (this is a common rule in TimeBank annotations).

We evaluate on the AQUAINT corpus, the TimeBank corpus, the Timen corpus (Llorens et al., 2012) and the TempEval 2013 test set (UzZaman et al., 2013)[5]. We compare to two[6] state-of-the-art systems: TIMEN and HeidelTime. Table 2 shows the results. Our synchronous grammar approach outperformed HeidelTime on all corpora, both on the training corpora (AQUAINT and TimeBank) and on the test corpora (Timen and TempEval 2013). Both our model and HeidelTime outperformed TIMEN on all corpora except for the Timen corpus.

To better understand the issues in the Timen corpus, we manually inspected the 33 time expressions that TIMEN normalized correctly and our approach normalized incorrectly. 4 errors were places where our heuristic was wrong (e.g. we chose the earlier, not the later *Sept. 22*). 6 errors were coverage problems of our grammar, e.g. not handling *season*, *every time* or *long ago*. 2 errors were actually human annotation errors (*several years ago* was annotated as PASTREF and *daily* was annotated as XXXX-XX-XX, while the guidelines say these should be PXY and P1D respectively). The remaining 21 errors were from two new normalization forms not present at all in the training data: 19 instances of THH:MM:SS (times were always YYYY-MM-DDTHH:MM:SS in the training data) and 2 instances of BCYYYY (years were always YYYY in the training data).

7 Discussion

Our synchronous grammar approach to time normalization, which handles recursive structures better than existing string-to-string approaches and handles a wider variety of grammars than existing parsing approaches, outperforms the HeidelTime system on four evaluation corpora and outperforms the TIMEN system on three of the four corpora.

Our time normalization code and models are freely available. The source code and English grammar are hosted at `https://github.com/bethard/timenorm`, and official releases are published to Maven Central (group=`info.bethard`, artifact=`timenorm`).

In future work, we plan to replace the heuristic for selecting between ambiguous parses with a more principled approach. It would be a simple extension to support a probabilistic grammar, as in (Angeli et al., 2012). But given an expression like *Monday*, it would still be impossible to decide whether it refers to the future or the past, since the surrounding context, e.g. tense of the governing verb, is needed for such a judgment. A more promising approach would be to train a classifier that selects between the ambiguous parses based on features of the surrounding context.

Acknowledgements

The project described was supported in part by Grant Number R01LM010090 from the National Library Of Medicine. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Library of Medicine or the National Institutes of Health.

[5]We evaluate normalization accuracy over all time expressions, not the F1 of both finding and normalizing expressions, so the numbers here are not directly comparable to those reported by the TempEval 2013 evaluation.

[6]Though its performance was slightly lower than HeidelTime, we also intended to compare to the (Angeli et al., 2012) system. Its authors graciously helped us get the code running, but to date all models we were able to train performed substantially worse than their reported results, so we do not compare to them here.

References

[Angeli et al.2012] Gabor Angeli, Christopher Manning, and Daniel Jurafsky. 2012. Parsing time: Learning to interpret time expressions. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 446–455, Montréal, Canada, June. Association for Computational Linguistics.

[Chappelier and Rajman1998] Jean-Cédric Chappelier and Martin Rajman. 1998. A generalized CYK algorithm for parsing stochastic CFG. In *First Workshop on Tabulation in Parsing and Deduction (TAPD98)*, pages 133–137. Citeseer.

[Llorens et al.2012] Hector Llorens, Leon Derczynski, Robert Gaizauskas, and Estela Saquete. 2012. TIMEN: An open temporal expression normalisation resource. In *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey, May. European Language Resources Association (ELRA).

[Pustejovsky et al.2003] James Pustejovsky, Patrick Hanks, Roser Sauri, Andrew See, Robert Gaizauskas, Andrea Setzer, Dragomir Radev, Beth Sundheim, David Day, Lisa Ferro, and Marcia Lazo. 2003. The TIMEBANK corpus. In *Corpus Linguistics*, pages 647–656, Lancaster, UK.

[Pustejovsky et al.2010] James Pustejovsky, Kiyong Lee, Harry Bunt, and Laurent Romary. 2010. ISO-TimeML: An international standard for semantic annotation. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, Valletta, Malta, may. European Language Resources Association (ELRA).

[Strötgen and Gertz2012] Jannik Strötgen and Michael Gertz. 2012. Multilingual and cross-domain temporal tagging. *Language Resources and Evaluation*.

[UzZaman et al.2013] Naushad UzZaman, Hector Llorens, Leon Derczynski, James Allen, Marc Verhagen, and James Pustejovsky. 2013. Semeval-2013 task 1: Tempeval-3: Evaluating time expressions, events, and temporal relations. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013)*, pages 1–9, Atlanta, Georgia, USA, June. Association for Computational Linguistics.

[Verhagen et al.2010] Marc Verhagen, Roser Sauri, Tommaso Caselli, and James Pustejovsky. 2010. SemEval-2010 task 13: TempEval-2. In *Proceedings of the 5th International Workshop on Semantic Evaluation*, page 5762, Uppsala, Sweden, July. Association for Computational Linguistics.