# Learning a Context-Dependent Semantic Parser for Temporal Expression Resolution

Jesse Dodge

Computer Science & Engineering
University of Washington

June 16, 2013

## Abstract

We present a semantic parsing approach for temporal expression resolution. Previous work, including current state of the art approaches, primarily use pattern matching and regular expression-like rules for grounding temporal expressions. However, the compositionality of language suggests a more linguistically principled approach.

In our system, using a hand-built lexicon and CCG parser, we first construct a base CCG parse for each temporal phrase, then build a set of candidate context-dependent logical forms from each base parse. We learn to select one parse from this set using a perceptron and features based on the logic, the phrase itself, and, critically, the full context the in which the phrase appears. Finally, we execute this logical form to a standardized date and time representation. This approach builds on previous successes using CCG for semantic parsing, and using a small featureset of linguistically motivated features, we achieve moderate accuracy on task A of TempEval3.

# 1 Introduction

Temporal expression resolution is the task of mapping from a temporal expression in text to a ground calendar date, set of dates, or duration. Temporal phrases are ubiquitous. They appear in all kinds of text, from e-mail to newswire to Wikipedia articles. Much of the information in print and Internet media consists of text that describes events occurring at a specific point in time. For a computer to be able to comprehend this text, it will need to be able to recognize events, and reason about how they are located temporally. Building a system that can understand temporal relations in text is a first step toward a system that can understand much of the information available in news articles, scientific journals and on the Internet.

Improved reasoning about times and events will enable us to build more effective systems for question answering, information extraction, and summarization. For example, a system that had extracted the two relations CEO(Steve Jobs, Apple) and CEO(Tim Cook, Apple) wouldnt be able to answer the question Who is the CEO of Apple? It would also need to understand the temporal relationship between the two relations.

## 1.1 Outline

In section 2 we cover previous work on this problem. Then, in section 3, we go over our representation and approach to parsing. Section 4 reveals our learning. Section 5 shows our results. Finally, section 6 talks about future work.

# 2 Previous Work

Previous work on this task has almost entirely been rule-based [3] [2]. The current state of the art approaches use a combination of regular-expression matching and hand-written interpretation functions to ground temporal expressions. These hand-built approaches are very domain-specific; they have a difficult time when applied to domains other than the one they were built for. Moreover, the complexities of natural language are often lost on these systems. Nested or hierarchical expressions are easy to handle when taking a parsing approach designed to deal with these phenomena, but these types of phrases can cause more rigid rule-based systems to fail. For example, phrases such as *the third Wednesday of each month* or *the second month of last year* are easier to ground using a parsing approach.

While most of the work in this area has been on deterministic systems, one notable exception is ParsingTime [1]. This system learned to parse from temporal phrases to logical forms, but is limited in that it doesn't take context into account. Their system uses a PCFG to parse the phrases to a logical form, then executes the form to get a result. However, their approach only looks at the

temporal phrase itsef; it doesn't have any signal from the context in which the phrase appeared, such as verb tense or the result of parsing previously uttered phrases.

Previous work has shown that using CCG for semantic parsing can be successful. Zettlemoyer and Collins [4] used CCG to map from text queries to logical forms, achieving state-of-the-art results on a number of datasets. In much of the previous work using CCG, the parse itself is used as a query to a database, which can be evaluated directly. In this work, however, the parse is latent; we execute the parse using a deterministic algorithm, as our signal comes from the fully ground date or duration. Other approaches have used a learning approach for building the lexicon used to parse. In this work, we are using a hand-built lexicon, for greater coverage and to enable future work in building a joint model of detection and parsing.

### 2.1 TempEval

Within SemEval, TempEval-2 and, more recently, TempEval-3 were competitions held that included a task on grounding temporal expressions. The dataset they released includes gold extents, types and values for the temporal phrases in a set of documents. Within this dataset there are four types, *1*) `date` (such as `1987-07-05` or `2013-Q3`), *2*) `time` (simply a date with a time, such as `1987-07-05T9`, for *9 am July 5th, 1987*), *3*) `set` (such as `XXXX-QX`, for *every quarter*), and *4*) `duration` (such as `P100Y`, for *one hundred years*).The values are more varied, as they represent more of the information the phrase grounds to. For example, the phrase *July 5th, 1987* grounds to the value `1987-07-05`.

This paper and the results section deal with the dataset from the task A of the TempEval competitions.

## 3 Representation

In this section, we first define our type system, then discuss our lexicon, then describe how we parse. Our system takes a three-step parsing approach. First, we use a CCG parser to build a set base parses for each temporal phrase in isolation (where each base parse is referred to a a context-independent parse). Then, for each context-independent parse, we deterministically build five context-dependent parses. Once we select a single parse from the set of candidate context-depndent parses (a process which we discuss in section **??**), we execute the parse to ground to a final representation.

### 3.1 Types within CCG

We define five types in our representation below.

**Definition 1** (Range)**.** A range is a period between two times. For example, *June 13th, 2013*, *today*, and *1987* all ground to ranges. Ranges also include general past and future references, such as the resolution of the phrase *in the future* or *previously*.


Figure 1: Above we can see three example parses. Each of these parses represents a base parse for the given temporal phrase. We can see how the base parse is built from the individual categories drawn from the lexicon. In the first example, we are parsing the phrase *third quarter*. The entries in the lexicon that are being used in this parse are $\lambda x.nth(x,3)$ and *quarter*, which correspond to *third* and *quarter*, respectively. These combine to create the base parse for the first phrase, $nth(quarter,3)$. The second example is very similar to the frist. However, the third example first does forward composition (combining $\lambda x*(x,1)$ with *year*) then does backward composition (combining $\lambda x.*(x,-1)$ with $*(year,1)$). The first context-independent parse, $nth(quarter,3)$ represents the sequence of all third quarters, which needs additional logic to be correctly ground to a single third quarter. The second parse, $*(weeks,5)$, can be fully ground to a duration of five weeks without needing an additional context step. The third parse, $*(*(year,1),-1)$, can't actually be ground to a date at all without additional contextual information.

**Definition 2** (Sequence)**.** A sequence represents an infinite sequence of ranges. For example, *each Thursday* grounds to a range. The duration between each range in a sequence is the same. In the example *each Thursday*, there is a week between each range. Sequences are represented as under-specified ranges. If we were interested in grounding the phrase *June 13th, 2013*, we first ground *June 13th* to the sequence of all June 13ths, which we could represent as XXXX-6-13, i.e. June 13th of an unspecified year, then intersect that sequence with the range 2013.

**Definition 3** (Duration)**.** A duration is a period of time with no specified start or end dates. For example, *two months*, *three years*, and *one day* all ground to durations

| Function | Description | Type |
|---|---|---|
| intersect | Finds the intersection of two sequences | $f:s*->s$ |
| * | Multiplies a duration by a number | $f:d,n->d$ |
| nth x of y | Finds the *nth* sequence within y (quarter of year, etc) | $f:d,n->s$ |
| next | Returns the range immediately following the document time | $f:s,r->s; f:d,r->s$ |
| this | Returns the range within which the document time falls | $f:s,r->s; f:d,r->s$ |
| previous | Returns the range immediately before the document time | $f:s,r->s; f:d,r->s$ |
| temproalReference | Grounds duration/sequence based on previous temporal phrase | $f:s->s; f:d->s$ |

Table 1: A list of all of the predicates in our logic of airity greater than zero.

| Type | Examples |
|---|---|
| Range | 1987, 2013-06-15, Past, 1990-Q2, etc. |
| Sequence | Friday, third quarter, each week, etc. |
| Duration | Hour, Day, Week, Month, etc. |

Table 2: A list of some examples for each of the three primary zero-arity predicates.

of time.

**Definition 4** (Number)**.** Numbers only appear in logical forms, never in fully executed output. Numbers are necessary when grounding durations, such as *4 years*, *1 hour*, or *3 weeks*.

**Definition 5** (Functional Types)**.** There are a number of functional types, all of arity less than or equal to two. The complete set is listed in 1.

## 3.2 Lexicon

A CCG is defined by a lexicon and set of combinators. In this work, we use standard forward and backward application with a hand-built lexicon to build our base parses. See 1 for an example.

## 3.3 Pasing Using CCG

First, for each temporal phrase in our dataset, we use a CKY parser to get a set of CCG logical forms. These are built only from the phrase itself, and don't take any additional information into account. Some ranges can be fully ground without looking at the context in which they appear, such as *June 6th, 2013*. Similarly, durations such as *five weeks* or *a year* don't need to look at surrounding context to be executed to a final form. However, the majority of phrases do require some context to ground fully. Phrases such as *Friday* are parsed to logic that represent a sequences, in this case the sequence of all Fridays. To ground phrases such as those to a single range, we need to look at context.

## 3.4 Building Context-Dependent Parse

From each context-independent logical form we get from the base CCG parser, we use a deterministic process to build a set of five context-dependent logical forms. These represent the five possible groundings of each of our base parses.

For fully-sepcified ranges and for durations, we don't actually need to change the logical form, so the frist of the the context-dependent logical forms is an unchanged version of the context-independent logical form.

Sequences often represent an infinite number of ranges, such as $nth(quarter, 3)$. We can ground these sequences to one of three ranges: The range before the document was published, the range after the document was published, or the range during which the document was published. These three groundings are represented by the next three logical forms, which are constructed by applying the three predicates $\lambda x.last(x, publicationTime)$, $\lambda x.this(x, publicationTime)$, and $\lambda x.next(x, publicationTime)$ to the base parse. We make the assumption that if someone were referring to a range other than the one before, during, or after the document was published that they would refer to it using a fully-specified range, instead of requiring resolution through context. This assumption allows us to only need these three additional context-dependent logical forms to ground any sequence, and matches our intuitions about how language is used.

Finally, for a class of temporal phrases that refer to a previously uttered temporal phrase, such as *a year earlier*, we build a logical form that grounds the current phrase based on the previous one by applying the predicate $\lambda x.temporalReference(x)$ to the base parse. An example of this process is shown in 2.

## 3.5 Execution

Once we have built a set of candidate context-dependent logical forms, we then execute them to fully ground the temporal phrases. Our execution is a deterministic process, done by walking the logic recursively. Within our logical forms, the no-argument predicates ground to a date or duration directly, while the predicates that take arguments act as functions over those ground dates or durations. Evaluating the logic recursively, we start with the outermost predicate and work our way in.

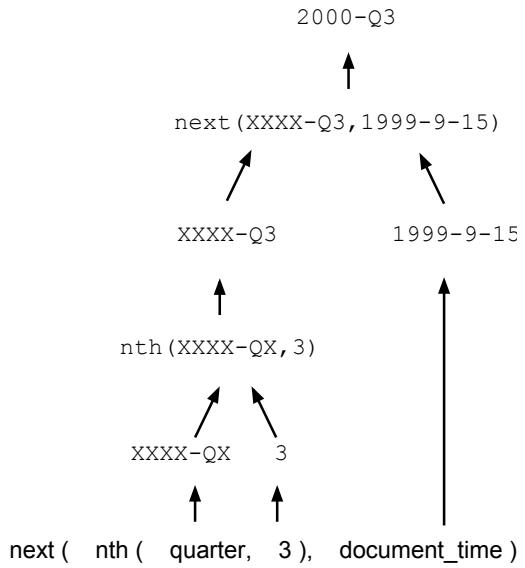


Figure 3: An example execution from a context-dependent logical form to a fully ground date, built from the phrase *third quarter*. Assumes the document the phrase appeared in was published September 15th, 1999. We recursively execute the logic `next(nth(quarter,3),documentTime)` to the final *value* `2000-Q3`. To do this, first we ground the zero-arity predicates, `quarter`, `3`, and `documentTime`. Then, we use those to ground the predicates with airity greater than zero, `nth` and `next`.

# 4 Learning

We now describe the learning approach used, including the algorithm and features. For this work, we use an online, error-driven perceptron. A training instance consists of *a*) the document (sentences, document publication date), *b*) the grounding for the previously uttered temproal phrase, *c*) the current temporal phrase, and *d*) the pair of the gold type and value.

First, we parse the current sentence, then chose the optimal logical form and execute it to check against our gold type and value. If the execution doesn't result in the correct type and value, we find the best logical form that does and run an additive, perceptron-style parameter update.

**Inputs:**

Training examples $I_i | i = 1 \ldots n$. Each $I_i$ is a sequence $d_j, p_i, c_i, tv_i$, where $d_j$ is the document for $I_i$, $p_i$ is the result of grounding $I_{i-1}$, $c_i$ is the current phrase, and $tv_i$ is the gold type and value pair. Number of training iteraniots $T$. Initial parameters $\theta$

**Definitions:**

The function $\phi(d)$ represents the features described in section 4.1. $GEN(w)$ is the set of derivations for phrase $w$. $GEN(w,tv)$ is the set of derivations for phrase $w$ that produce a logical form that executes to type and value pair $tv$. The function $L(d)$ maps a derivation to its associated final executed output.

**Algorithm:**

- For $t = 1 \ldots T, i = 1 \ldots n$ : (Iterations)

  **Step 1:** (check correctness)
  - Let $d* = argmax_{d \in GEN(w_i)} \theta * \phi(d)$.
  - If $L(d*) = tv_i$, go to Step 3.

  **Step 2:** (Update parameters)
  - Let $d' = argmax_{d \in GEN(w_i, tv_i)} \theta * \phi(d)$.
  - Set $\theta = \theta + \phi(d') - \phi(d*)$.

**Output:** Estimated parameters $\theta$.

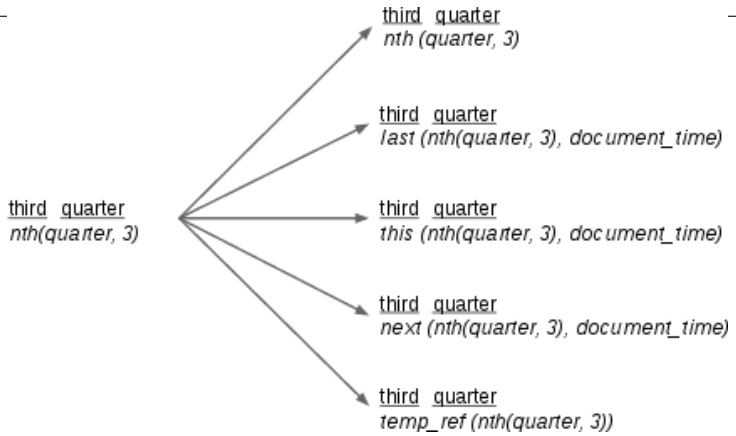


Figure 2: The creation of the five context-dependent logical forms from one context-independent logical form. This determinisitic process always generates five logical forms by applying five predicates to the base parse. First, a predicate $\lambda x.x$, which always returns its argument. Next, three predicates to ground a single range out of a sequence, $\lambda x.last(x, documentTime)$, $\lambda x.this(x, documentTime)$, and $\lambda x.next(x, documentTime)$. Finally, a predicate $\lambda x.temporalReference(x)$ that resolves $x$ based on the previously uttered temproal phrase.

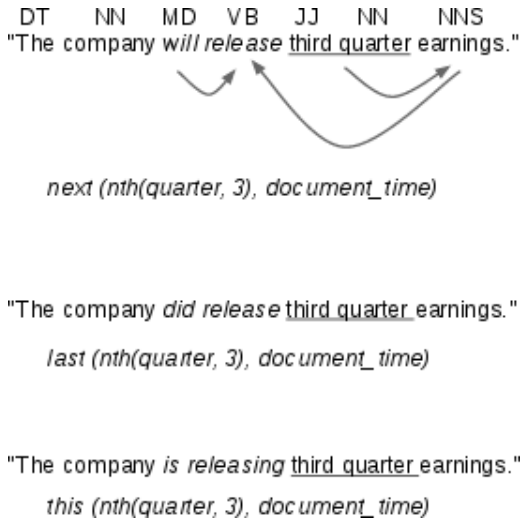


Figure 4: When grounding the phrase *third quarter* in the example sentences above, we compute features based on the tense of the verb that governs *third quarter*. These features help us select between the three context-dependent logical forms shown above.

### 4.1 Features

We use a small set of linguistically motivated features for this work. We have a set of lexical features over the base parse, which are sensitive to the lexical choices and the structure of the logical form that is constructed. We also have a set of features derived from the context in which a given temporal phrase appears. These features use a dependency parse to find the verb that governs the temporal phrase, then look at its POS tag and the POS tag of any other verbs paired with it via a dependency arc. These features represent information on the tense of the govener verb, and help us choose between the different context-dependent logical forms. (See 4.) Finally, we have an indicator feature for phrases that need to be ground relative to the previously uttered temporal phrase, such as *a year earlier* or *that quarter*.

## 5 Results

Here we compare our system to three rule-based systems, including the current state of the art system, and one other learning system. While the training type and value are only slightly below the other systems, the test type and value are significantly below.

| | **Train** | | **Test** | |
|---|---|---|---|---|
| **System** | **Type** | **Value** | **Type** | **Value** |
| GUTime | .72 | .46 | .8 | .42 |
| SUTime | .85 | .69 | .94 | .71 |
| HeidelTime | .8 | .76 | .85 | .71 |
| ParsingTime | .90 | .72 | .88 | .72 |
| My system | .82 | .66 | .79 | .57 |

## 6 FutureWork

In the near future, there is still work to be done on a number of aspects of our system. The current lexicon only covers about 90% of the dataset, meaning about 10% of the phrases don't have any parses. If a given phrase does parse, however, our systems still has trouble selecting the correct context-dependent logical form. This can be seen by the discrepancy between our training and testing accuracies. The low testing accuracy is because the model we learn doesn't generalize well. Examining the errors will shed light on how we can adjust our feature set to better learn a model of our data.

Within this dataset, each of the temporal phrases are annotated with gold mentions. This system uses these mentions, but in the future learning to detect these phrases will be vital. As we have a hand-built lexicon, we can use that lexicon for both detection and parsing by trying to parse phrases within documents.

Work in this area is a good first-step towards computational understanding, but expanding our reasoning about times to include reasoning about the events occuring at those times would be very beneficial. This would be a step toward building systems that can better perform question answering, information extraction, and summarization.

While the vast majority of natural lanuage research has been done on English text, temporal phrases appear in every language. Even in the TempEval tasks, there are datasets for six languages that have annotated temporal phrases. Expanding the system to work in a number of languages would only require rewriting the lexicon, but will be left to future projects.

## References

[1] ANGELI, G., MANNING, C. D., AND JURAFSKY, D. Parsing time: Learning to interpret time expressions. In *North American Chapter of the Association for Computational Linguistics (NAACL)* (2012).

[2] CHANGE, A., AND MANNING, C. D. Sutime: a library for recognizing and normalizing time expressions. In *Language Resources and Evaluation*.

[3] STROTGEN, J., AND GERTZ, M. Heideltime: High quality rule-based extraction and normalization of temporal expressions. In *Proceedings of the 5th International Workshop on Semantic Evaluation*, pp. 321–324.

[4] ZETTLEMOYER, L., AND COLLINS, M. Learning to map sentences to logical form:structured classification with probabilistic categorial grammars. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence* (2005).