Journal of Artificial Intelligence Research 48 (2013) 305-346 Submitted 04/13; published 10/13

A Global Model for Concept-to-Text Generation

Ioannis Konstas IKONSTAS@INF.ED.AC.UK
Mirella Lapata MLAP@INF.ED.AC.UK
Institute for Language, Cognition and Computation,
School of Informatics, University of Edinburgh,
10 Crichton Street, EH8 9AB, Edinburgh UK

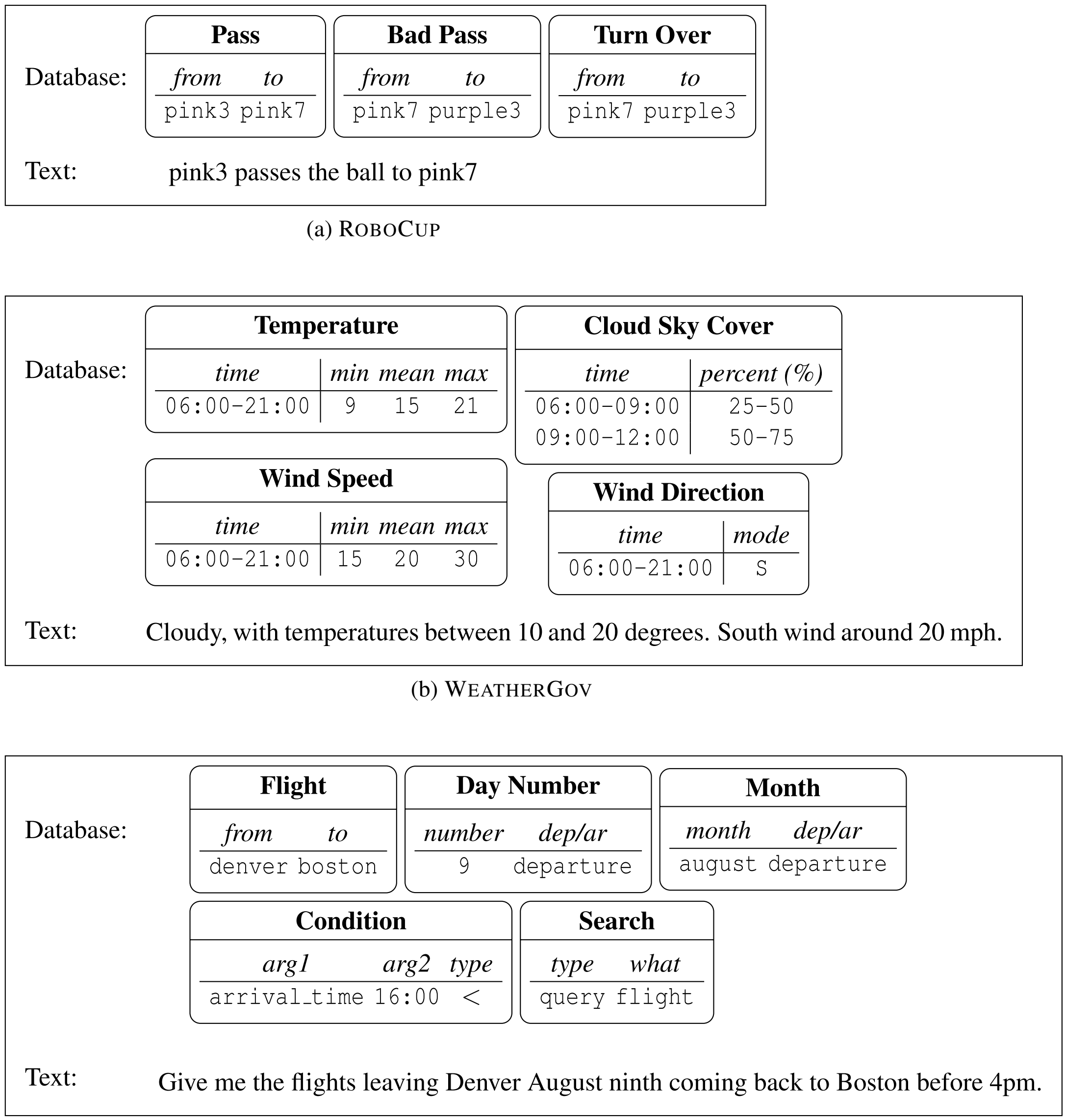
Abstract

Concept-to-text generation refers to the task of automatically producing textual output from non-linguistic input. We present a joint model that captures content selection ("what to say") and surface realization ("how to say") in an unsupervised domain-independent fashion. Rather than breaking up the generation process into a sequence of local decisions, we define a probabilistic context-free grammar that globally describes the inherent structure of the input (a corpus of database records and text describing some of them). We recast generation as the task of finding the best derivation tree for a set of database records and describe an algorithm for decoding in this framework that allows to intersect the grammar with additional information capturing fluency and syntactic well-formedness constraints. Experimental evaluation on several domains achieves results competitive with state-of-the-art systems that use domain specific constraints, explicit feature engineering or labeled data.

1. Introduction

Concept-to-text generation broadly refers to the task of automatically producing textual output from non-linguistic input (Reiter & Dale, 2000). Depending on the application and the domain at hand, the input may assume various representations including databases of records, expert system knowledge bases, simulations of physical systems and so on. Figure 1 shows input examples and their corresponding text for three domains: air travel, sportscasting and weather forecast generation.

A typical concept-to-text generation system implements a pipeline architecture consisting of three core stages, namely content planning (selecting the appropriate content from the input and determining the structure of the target text), sentence planning (determining the structure and lexical content of individual sentences), and surface realization (rendering the specification chosen by the sentence planner into a surface string). Traditionally, these components are hand-engineered in order to generate high quality text, at the expense of portability and scalability. It is thus no surprise that recent years have witnessed a growing interest in automatic methods for creating trainable generation components. Examples include learning which database records should be present in a text (Duboue & McKeown, 2002; Barzilay & Lapata, 2005) and how these should be verbalized (Liang, Jordan, & Klein, 2009). Besides concentrating on isolated components, a few approaches have emerged that tackle concept-to-text generation end-to-end. Due to the complexity of the task, most models simplify the generation process, e.g., by creating output that consists of a few sentences, thus obviating the need for content planning, or by treating sentence planning and surface realization as one component. A common modeling strategy is to break up the generation process into a sequence of local decisions, each learned separately (Reiter, Sripada, Hunter, & Davy, 2005a; Belz, 2008; Chen & Mooney, 2008; Angeli, Liang, & Klein, 2010; Kim & Mooney, 2010).



(a) ROBOCUP

(b) WEATHERGOV

(c) ATIS

Figure 1: Input-output examples for (a) sportscasting, (b) weather forecast generation, and (c) query generation in the air travel domain.

In this paper we focus on the problem of generating text from a database and describe an end-to-end generation model which performs content selection and surface realization jointly. More specifically, the input to our model is a set of database records and collocated textual descriptions. Consider the example in Figure 1b. Here, the records provide a structured representation of the weather for a specific time interval (e.g., the temperature, the wind speed and direction) and the text renders some of this information in natural language. We formulate the task of creating text corresponding to a database through the following generative process: the database consists of a set of typed tuples (*record, field, value*), and our aim is to choose a subset of these to talk about. This naturally decomposes into selecting a sequence of records, and a sequence of fields within the record. Finally, for each field we generate a sequence of words according to the value of that field. Central to our approach is to jointly optimize this process, rather than breaking up the various decisions into local problems and greedily trying to solve each one of them.

To do this, we define a probabilistic context-free grammar (PCFG) that captures the structure of the database and how it can be verbalized. Generation then boils down to finding the best string output as captured by the best derivation tree licensed by our grammar. In order to ensure that our generation output is coherent, we intersect our grammar with additional information capturing fluency and syntactic well-formedness constraints. Specifically, we experiment with a n-gram language model and a dependency model based on the work of Klein and Manning (2004). We follow Chiang's (2007) integration framework and show how it can be extended by intersecting a CFG grammar with an arbitrary number of models (see Huang, 2008 for a similar proposal). Our work is closest to that of Liang et al. (2009) who learn how to align database records and text segments using a hierarchical hidden semi-Markov generative model (see Section 3.1 for details). We recast their model as a PCFG and develop a decoding algorithm that allows us to go beyond alignments, i.e., to generate multi-sentence text corresponding to database input.

Our model is conceptually simpler than previous approaches (e.g., Angeli et al., 2010; Kim & Mooney, 2010); it encodes information about the domain and its structure globally, by considering the input space *simultaneously* during generation. We thus need to train a single model (on a given domain) once without having to separately optimize different content selection and surface realization components. More importantly, recasting generation into parsing allows us to optimize a joint objective (hence finding the most likely grammar derivation that *also* yields a grammatical output text) in a more principled manner, rather than approximating it with a greedy search over local decisions. Our only assumption is that the input must be a set of records essentially corresponding to database-like tables whose columns describe fields of a certain type. Experimental evaluation on three domains obtains results competitive to the state-of-the-art without using any domain specific constraints, explicit feature engineering or labeled data.[1]

The remainder of this paper is structured as follows. Section 2 provides an overview of related work. Section 3 presents our generation model; it defines the PCFG used in our experiments and presents our decoding algorithm Section 4 discusses our experimental set-up and Section 5 presents our results. Discussion of future work concludes the paper.

2. Related Work

The literature reveals many examples of generation systems that produce high quality text, almost indistinguishable from human writing (Dale, Geldof, & Prost, 2003; Reiter, Sripada, Hunter, Yu, & Davy, 2005b; Green, 2006; Turner, Sripada, & Reiter, 2009). Such systems often implement a pipeline architecture and involve a great deal of manual effort. For instance, a typical content selection module involves manually engineered rules based on the analysis of a large number of texts from a domain-relevant corpus, and consultation with domain experts. Analogously, surface

1. A preliminary version of this work was published in the proceedings of NAACL 2012. The current article presents a more general model, formulates explicitly our decoding algorithm and shows how to intersect a PCFG with an arbitrary number of external knowledge sources. In addition, we present several novel experiments, and a comprehensive error analysis.

realization is often based on a grammar written by hand so as to cover the syntactic constructs and vocabulary of the domain.

One of the earliest systems that exemplifies this approach is FOG (Goldberg, Driedger, & Kittredge, 1994), a weather forecast generator used by Environment Canada, the Canadian weather service. FOG takes as input numerical simulations from meteorological maps and uses an expert system to decide on the structure of the document with some optional human intervention via a graphical interface. For sentence planning and surface realization, the generator uses a grammar specific to the weather domain, as well as canned syntactic structures written by expert linguists and encoded in Backus Naur Form (BNF). More recently, Reiter et al. (2005a) have developed SUMTIME-MOUSAM, a text generator that produces marine weather forecasts for offshore oil-rig applications. The content planner of the system is based on linear segmentation of the input (i.e., time series data) and is informed by a pragmatic (Gricean) analysis of what should be communicated in weather forecasts (Sripada, Reiter, Hunter, & Yu, 2003). Sentence planning relies on rules that select appropriate time phrases, based on an empirical study of human-written forecasts. Surface realization relies on special grammar rules that emulate the weather sub-language of interest, again based on corpus analysis.

While existing generation systems can be engineered to obtain good performance on particular domains, it is often difficult to adapt them across different domains. An alternative is to adopt a data-driven approach and try to automatically learn the individual generation components or even an end-to-end system. An example of this class of methods is described in the work of Barzilay and Lapata (2005) who view content selection as an instance of collective classification. Given a corpus of database records and texts describing some of them, they first use a simple anchor-based alignment technique to obtain records-to-text alignments. Then, they use the alignments as training data (records present in the text are positive labels, and all other records negative) and learn a content selection model that simultaneously optimizes local label assignments and their pairwise relations. Building on this work, Liang et al. (2009) present a hierarchical hidden semi-Markov generative model that first determines which facts to discuss and then generates words from the predicates and arguments of the chosen facts. Their model is decomposed into three tiers of HMMs that correspond to chains of records, fields and words. They use Expectation Maximization (EM) for training and dynamic programming for inference (see Section 3.1 for a more thorough description).

A few approaches have emerged more recently that combine content selection and surface realization. Kim and Mooney (2010) present a generator with a two-stage pipeline architecture: using a generative model similar to the model in the work of Liang et al. (2009), they first decide what to say and then verbalize the selected input with WASP^{-1}, an existing generation system (Wong & Mooney, 2007). In contrast, Angeli et al. (2010) propose a unified content selection and surface realization model which also operates over the alignment output produced by the model of Liang et al.. Their model decomposes into a sequence of discriminative local decisions. They first determine which records in the database to talk about, then which fields of those records to mention, and finally which words to use to describe the chosen fields. Each of these decisions is implemented as a log-linear model with features learned from training data. Their surface realization component performs decisions based on automatically extracted templates that are filtered with domain-specific constraints in order to guarantee fluent output.

Other related work has focused on mapping meaning representations (e.g., some logical form or numeric weather data) to natural language, using explicitly aligned sentence/meaning pairs as training data. For example, Wong and Mooney (2007) learn this mapping using a synchronous context-free grammar (SCFG). They also integrate a language model with their SCFG and decode the meaning representation input to text, using a left-to-right Early chart generator. Belz (2008) creates a CFG by hand (using a set of template-based domain-specific rules) but estimates probabilities for rule application automatically from a development corpus. Ratnaparkhi (2002) uses a dependency-style grammar of phrase fragments in the context of a dialogue system, incorporating among others long-range dependencies. More recently, Lu and Ng (2011) propose in their work a model that performs joint surface realization and lexical acquisition from input that is represented in typed lambda calculus. They present a novel SCFG forest-to-string generation algorithm, that captures the correspondence between natural language and logical form represented by $\lambda-$hybrid trees.

Similar to the work of Angeli et al. (2010), we also present an end-to-end system that performs content selection and surface realization. However, rather than breaking up the generation task into a sequence of local decisions, we optimize what to say and how to say simultaneously. We do not learn mappings from a logical form, but rather focus on input which is less structured and possibly more noisy. Our key insight is to convert the set of database records serving as input to our generator into a PCFG that is neither hand crafted nor domain specific but simply describes the structure of the input. During training, we estimate the weights of the grammar rules using the EM algorithm and a dynamic program similar to the inside-outside algorithm (Li & Eisner, 2009). During testing we are given only a set of database and search for the best derivation tree licensed by the grammar. While searching, we intersect our grammar with external linguistically motivated models and create k-best lists of derivations, thus optimizing “what to say” and “how to say” at the same time.

3. Problem Formulation

We assume our generator takes as input a set of database record tuples $(r, f, v) \in \mathbf{d}$ and outputs a text g that verbalizes some of these records. Each record token r_i, with $1 \leq i \leq |\mathbf{d}|$, has a type $r_i.t$, which can be thought of as the name of the table in a relational database schema. Note that the total number of records $|\mathbf{d}|$ can vary between examples. Figure 1b illustrates instances of record types such as **Temperature**, **W**ind Speed, and **Wind Direction**. Each record token also has a set of fields $r_i.\mathbf{f}$ associated with it. For example, a record of type **Wind Direction** has two fields, namely windDir$_1$.*time* and windDir$_1$.*mode*. We will henceforth abbreviate fields to their names (e.g., *time* and *mode*) when the record type is apparent from the context. Fields have different values $f_k.v$; in Figure 1b the value of the field *mode* is `S`. Fields also have an associated type $f_k.t$, which defines the range of possible values they can take; our model supports integer and categorical value types. For example, the top right table in Figure 1b named **Cloud Sky Cover** (sc for short), corresponds to four database record tuples: (sc$_1$, *time*, `06:00-09:00`), (sc$_1$, *percent*, `25-50`), (sc$_2$, *time*, `09:00-12:00`) and (sc$_2$, *percent*, `50-75`). Both *time* and *percent* are of categorical type.

The training corpus consists of several *scenarios*, i.e., database records **d** paired with texts w[2] like those shown in Figure 1. In the weather forecast domain, a scenario corresponds to weather-related measurements of temperature, wind, speed, and so on collected for a specific day and time (e.g., day or night). In sportscasting, scenarios describe individual events in the soccer game (e.g., passing or kicking the ball). In the air travel domain, scenarios comprise of flight-related details (e.g., origin, destination, day, time).

2. We use w to denote the gold-standard text and g to refer to the string of words our system generates.

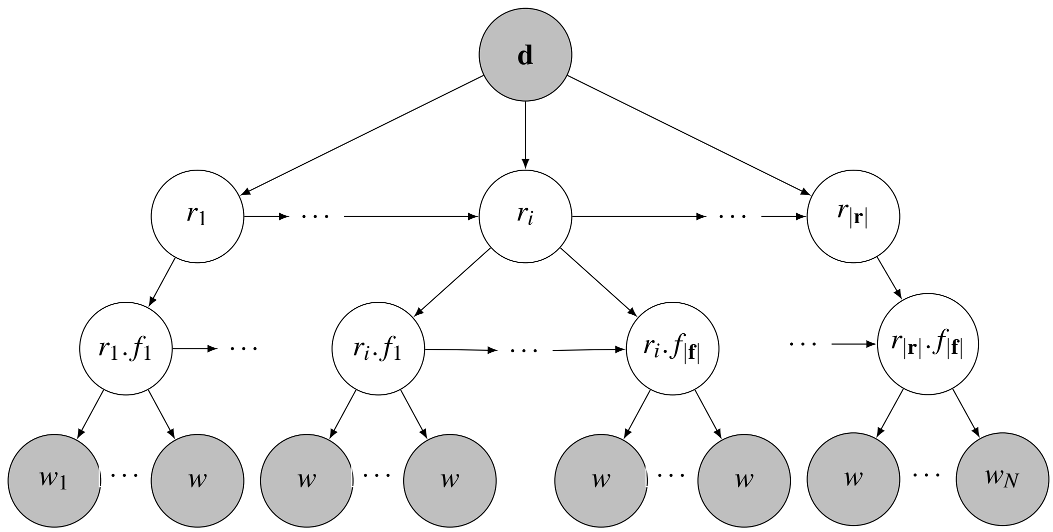


Figure 2: Graphical model representation of the generative alignment model of Liang et al. (2009). Shaded nodes represent observed variables (i.e., the database **d** and the collocated text w), unshaded nodes indicate latent variables. Arrows indicate conditional dependencies between variables. Starting from the database **d**, the model emits a sequence of records; then for each record it emits a sequence of fields, specific to the type of the particular record. Finally, for each record it uniformly selects a number c and emits words $w_1 \dots w_c$.

Our goal is to first define a model that naturally captures the (hidden) relations between the database records **d** and the observed text w. Once trained, we can use this model to generate text g corresponding to new records **d**. Our model is an extension of the hierarchical hidden semi-Markov model of Liang et al. (2009) which we describe in detail in the next section. Our key idea is to recast this model as a probabilistic context-free grammar, therefore reducing the tasks of content selection and surface realization into a common parsing problem.[3] Arguably, we could have implemented this model using a finite-state representation. However, the conceptualization of generation as parsing, allows us to use the well-known CYK algorithm (Kasami, 1965; Younger, 1967) in order to find the best g licensed by the grammar. It also affords a wider range of extensions that go beyond the expressivity of the cascade of HMMs in the model of Liang et al. We furthermore ensure that the resulting text is fluent by intersecting our grammar with externally trained surface level models, namely a n-gram language model and a dependency model. Thus, our model will generate the parse and more importantly text deemed most likely by both the grammar *and* the surface models. In the following, we first describe the approach of Liang et al. and then move on to describe our grammar and decoding algorithm, i.e., our procedure for finding the best g for a given input **d**.

3.1 A Model of Inducing Alignments

Liang et al. (2009) present a generative semi-hidden Markov model that learns the correspondence between a *world state* and an unsegmented string of text without, however, generating an output string of words g describing the world state. As in our case, the world state is represented by a set of database records, with their associated fields and values. Their model is defined by a generative process that can be summarized in three steps:

1. *Record choice.* Choose a sequence of records $\mathbf{r}$ to describe. Consecutive records are selected on the basis of their types.
2. *Field choice.* For each record r_i emit a sequence of fields $r_i.\mathbf{f}$.
3. *Word choice.* For each chosen field $r_i.f_k$ generate a number of words c, where $c > 0$ is chosen uniformly.

This process is implemented as a hierarchy of Markov chains which correspond to records, fields, and values of the input database. As captured by a Markov chain of records conditioned on record types; given a record type, then a record is chosen uniformly from the set of records with this type. In this way, their model essentially captures rudimentary notions of local coherence and salience, respectively. More formally:

$$p(\mathbf{r}\,|\,\mathbf{d}) = \prod_{i}^{|\mathbf{r}|} p(r_i.t\,|\,r_{i-1}.t)\frac{1}{|\mathbf{s}(r_i.t)|} \tag{1}$$

where $\mathbf{s}(t)$ is defined as a function that returns the set of records with type t: $\mathbf{s} = \{r \in \mathbf{d} : r.t = t\}$, and $r_0.t$ is the **START** record type. Liang et al. (2009) also include a special **null** record type, which accounts for words that do not particularly align with any record present in the database. Field choice is modeled analogously as a Markov chain of fields for a given record choice r_i of type t:

$$p(\mathbf{f}\,|\,r_i.t) = \prod_{k}^{|r_i.\mathbf{f}|} p(r_i.f_k\,|\,r_i.f_{k-1}) \tag{2}$$

They also implement special *start* and *stop* fields to model transitions at the boundaries of the corresponding phrase. Finally, for a chosen record r_i, a field f_k and a uniformly chosen number c, with $0 < c < N$, they emit words *independently* given the field value and type. Note that since their model always observes the words, this simplistic representation at the surface level is adequate (however, relaxing the independence assumption, e.g., by additionally conditioning on the previous word(s), could potentially yield a more powerful model):

$$p(\mathbf{w}\,|r_i, r_i.f_k, r_i.f_k.t) = \prod_{j}^{|\mathbf{w}|} p(w_j\,|\,r_i.t, r_i.f_k.v) \tag{3}$$

Their model supports three different types of fields, namely string, categorical and integer. For each of those they adopt a specific generation strategy at the word level. For string-typed fields, they

3. An alternative would be to learn a SFCG between the database input and the accompanying text. However, this would involve considerable overhead in terms of alignment (as the database and the text do not together constitute a clean parallel corpus, but rather a noisy comparable corpus), as well as grammar training and decoding using state-of-the art statistical machine translation (SMT) methods, which we manage to avoid with our simpler approach.

Events:	skyCover_1	temperature_1				windDir_1		windSpeed_1	
Fields:	*percent*=`0-25`		*time*=`6am-9pm`	*min*=`9`	*max*=`21`	*mode*=`S`			mean=`20`
Text:	cloudy ,	with	temperatures between	10	20 degrees .	south	wind	around	20 mph .

Figure 3: Example of alignment output for the model of Liang et al. (2009) on the weather domain. Subscripts refer to record tokens (e.g., skyCover_1 is the first record with type **Cloud Sky Cover**).

emit a single word from the (possibly) multi-word value, chosen uniformly. For categorical fields, they maintain a separate multinomial distribution of words for each field value. Finally, for integer fields, they wish to capture the intuition that a numeric quantity in the database can be rendered in the text as a word which is possibly some other numerical value due to stylistic factors. So they allow several ways generating a word given a field value. These include generating the exact value, rounding up or rounding down to a multiple of 5, rounding off to the closest multiple of 5, and adding or subtracting some unexplained noise ε_+ or ε_-, respectively. Each noise is modeled as a geometric distribution, the parameters of which are trained given the value $r_i.f_k.v$.

An example of the model's output for the weather domain is shown in Figure 3. The top row contains the database records selected by the model (subscripts correspond to record tokens; e.g., temperature_1 refers to the first record of type temperature in Figure 1b). The second row contains the selected fields for each record with their associated values. The special field *null* aligns with words that do not directly refer to the values of the database records, such as *with*, *wind* and *around*. Finally, the last row shows the segmentation and alignment of the original text w produced by the model.

As it stands, Liang et al.'s (2009) model generates an alignment between sequences of words and facts in a database, falling short of creating a meaningful sentence or document. Kim and Mooney (2010) address this problem by interfacing the alignments with WASP^{-1} (Wong & Mooney, 2007). The latter is a publicly available generation system which takes an alignment as input and finds the most likely string using the widely popular noisy-channel model. Angeli et al. (2010) propose a model different in spirit which nevertheless also operates over the alignments of Liang et al. Using a template extraction method, they post-process the alignments in order to obtain a sequence of records, fields, and words which are spanned by the chosen records and fields. The generation process is then modeled as a series of local decisions, arranged hierarchically and each trained discriminatively. For each record they choose to talk about, they then choose a subset of fields, and finally a suitable template to render the chosen content. The same process repeats until it decides to generate a special **STOP** record.

We do not treat the model of Liang et al. (2009) as a black box in order to obtain alignments. Rather, we demonstrate how generation can be seamlessly integrated in their semi-hidden Markov model by re-interpreting it as CFG rewrite rules and providing an appropriate decoding algorithm. Our model *simultaneously* learns which records and fields to talk about, which textual units they correspond to, and how to creatively rearrange them into a coherent document.

3.2 Grammar Definition

As mentioned earlier, we recast the model of Liang et al. (2009) as a series of CFG rewrite rules, corresponding to the first two layers of the HMMs in Figure 2. We also include a set of grammar rules that emit chains of words, rather than words in isolation. This can be viewed as an additional

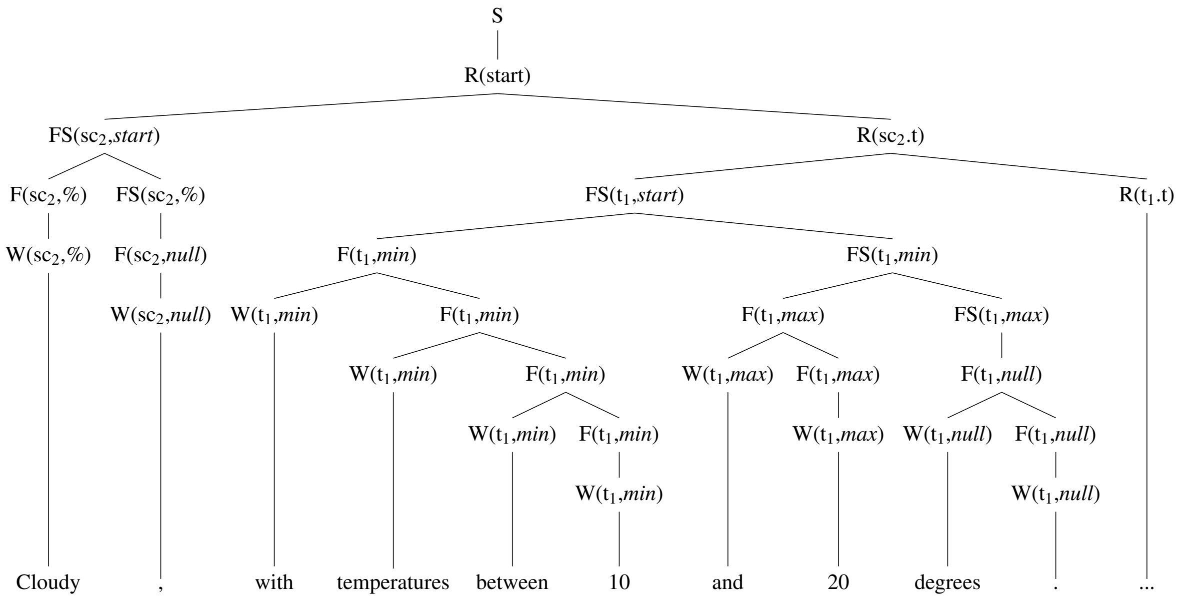
G_{CS}	1. $\mathrm{S} \rightarrow \mathrm{R}(\mathrm{start})$	$[Pr = 1]$
	2. $\mathrm{R}(r_i.t) \rightarrow \mathrm{FS}(r_j, start)\ \mathrm{R}(r_j.t)$	$\left[P(r_j.t \mid r_i.t) \cdot \frac{1}{\lvert \mathbf{s}(r_i.t) \rvert}\right]$
	3. $\mathrm{R}(r_i.t) \rightarrow \mathrm{FS}(r_j, start)$	$\left[P(r_j.t \mid r_i.t) \cdot \frac{1}{\lvert \mathbf{s}(r_i.t) \rvert}\right]$
	4. $\mathrm{FS}(r, r.f_i) \rightarrow \mathrm{F}(r, r.f_j)\ \mathrm{FS}(r, r.f_j)$	$[P(f_j \mid f_i)]$
	5. $\mathrm{FS}(r, r.f_i) \rightarrow \mathrm{F}(r, r.f_j)$	$[P(f_j \mid f_i)]$
	6. $\mathrm{F}(r, r.f) \rightarrow \mathrm{W}(r, r.f)\ \mathrm{F}(r, r.f)$	$[P(w \mid w_{-1}, r, r.f)]$
	7. $\mathrm{F}(r, r.f) \rightarrow \mathrm{W}(r, r.f)$	$[P(w \mid w_{-1}, r, r.f)]$
G_{SURF}	8. $\mathrm{W}(r, r.f) \rightarrow \alpha$	$[P(\alpha \mid r, r.f, f.t, f.v, f.t = \{cat, null\})]$
	9. $\mathrm{W}(r, r.f) \rightarrow \mathrm{gen}(f.v)$	$[P(\mathrm{gen}(f.v).mode \mid r, r.f, f.t = int) \cdot P(f.v \mid gen(f.v).mode)]$

Table 1: Grammar rules for G_{GEN} and their weights shown in square brackets.

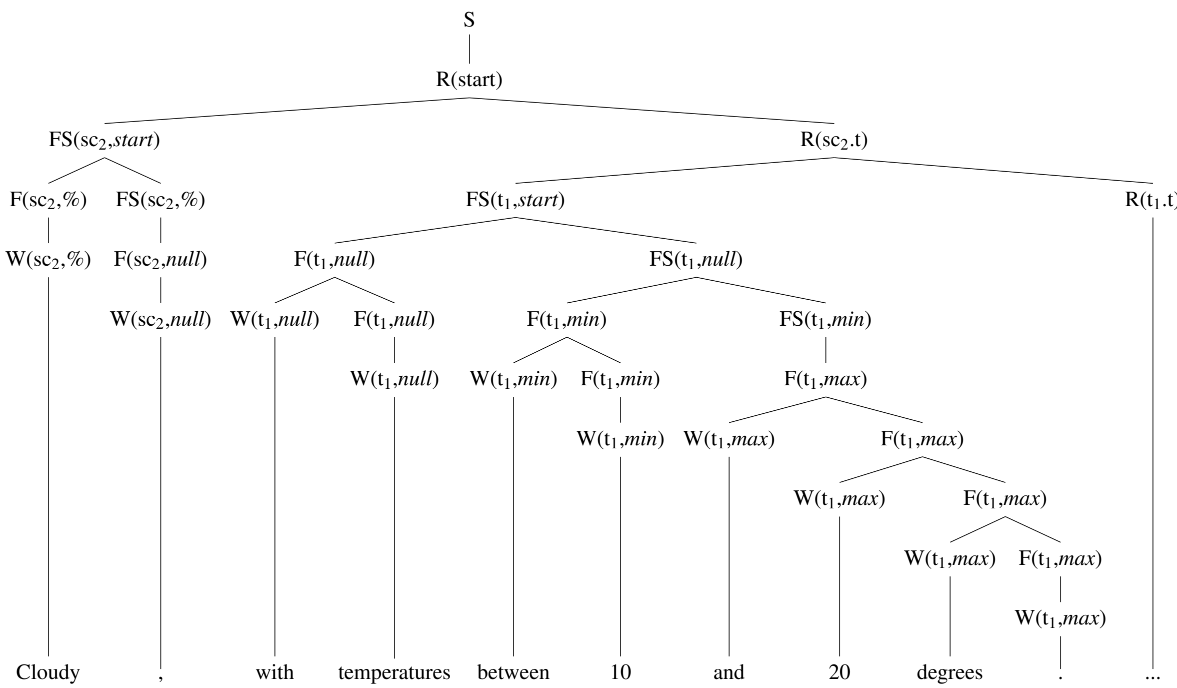
HMM over words for each field in the original model. The modification is important for generation; since we only observe the set of database records **d**, we need a better informed model during decoding that captures word-to-word dependencies more directly. We should also point out that our PCFG does not extend the underlying expressivity of the model presented in Liang et al., namely it also describes a regular language.

Our grammar G_{GEN} is defined in Table 1 (rules (1)–(9)) and contains two types of rules. G_{CS} rules perform content selection, whereas G_{SURF} rules perform surface realization. Both types of rules are purely syntactic (describing the intuitive relationship between records, records and fields, fields and corresponding words), and could apply to any database with similar structure irrespectively of the semantics of the domain. Rule weights are governed by an underlying multinomial distribution and are shown in square brackets. Non-terminal symbols are in capitals and denote intermediate states; the terminal symbol α corresponds to all words seen in the training set, and gen$(f.v)$ is a function for generating integer numbers given the value of a field f. All non-terminals, save the start symbol S, have one or more features (shown in parentheses) which act as constraints, similar to number and gender agreement constraints in augmented syntactic rules. Figure 4 shows two derivation trees licensed by our grammar for the sentence "*Cloudy, with temperatures between 10 and 20 degrees.*" (see the example in Figure 1b).

The first rule in the grammar denotes the expansion from the start symbol S to record R, which has the special 'start' record type (hence the notation R(start)). Rule (2) defines a chain between two consecutive records, i.e., going from record r_i to r_j. Here, $\mathrm{FS}(r_j, start)$ represents the set of fields of record r_j following record $\mathrm{R}(r_i)$. For example, in Figure 4a, the top branching rule $\mathrm{R}(\mathrm{start}) \rightarrow \mathrm{FS}(sc_2, start)\mathrm{R}(sc_2.t)$ (sc stands for **Cloud Sky Cover**) can be interpreted as follows. Given we are at the beginning of the document, hence the record R(start), we will talk about the



(a)



(b)

Figure 4: Two derivation trees using the grammar in Table 1 for the sentence "*Cloudy, with temperatures between 10 and 20 degrees.*". We use *sc* as a shorthand for the record type **Cloud Sky Cover**, and *t* for **Temperature**. Subscripts refer to record tokens (e.g., sc_2 is the second **Cloud Sky Cover** record, t_1 is the first Temperature record, and so on).

part of the forecast that refers to **Cloud Sky Cover**, i.e., emit the set of fields spanned by the non-terminal $\text{FS}(sc_2, start)$. The field *start* in FS acts as a special boundary between consecutive records. Note that in the input database of example 1b, there are two records of type **Cloud Sky Cover** (see the second box in the example). Given that the value of the *percent (%)* field of the second record is `50-75`, it is more likely to lexicalize to the phrase "*Cloudy ,*". In a different scenario, if the equivalent phrase was "*Mostly sunny ,*" the first record with value `25-50` would have been more appropriate. Rule $\text{R}(sc_2.t) \rightarrow \text{FS}(t_1, start)\text{R}(t_1.t)$ (t stands for **Temperature**) is interpreted similarly: once we talk about the sky coverage of the forecast we will move on to describe the temperature outlook, via the field set spanned by the non-terminal $\text{FS}(t_1, start)$ (see the second sub-tree in Figure 4a). The weight of this rule is the bigram probability of two records conditioned on their record type, multiplied with the normalization factor $\frac{1}{|\mathbf{s}(r_i.t)|}$, where $\mathbf{s}(t)$ is a function that returns the set of records with type t (Liang et al., 2009). We have also defined a **null** record type i.e., a record that has no fields and acts as a smoother for words that may not correspond to a particular record. Rule (3) is simply an escape rule, so that the parsing process (on the record level) can finish.

Rule (4) is the equivalent of rule (2) at the field level, i.e., it describes the chaining of two consecutive fields f_i and f_j. Non-terminal $\text{F}(r, r.f)$ refers to field f of record r. For example, in the tree of Figure 4a, the rule $\text{FS}(t_1, min) \rightarrow \text{F}(t_1, max)\ \text{FS}(t_1, max)$ specifies that we should talk about the field *max* of record t_1 (i.e., temperature record), after talking about the field *min*. Analogously to the record level, we have also included a special *null* field type for the emission of words that do not correspond to a specific record field (e.g., see the emission of the two last tokens "*degrees .*" in the end of the phrase in the derivation tree. Rule (6) defines the expansion of field F to a sequence of (binarized) words W, with a weight equal to the bigram probability of the current word given the previous word, the current record, and field. See the consecutive application of this rule on the derivation tree in the emission of the phrase "*with temperatures between 10*".

Rules (8) and (9) are responsible for surface generation; they define the emission of words and integers from W, given a field type and its value, and can thus be regarded as the lexical rules of our grammar (see the pre-terminal expansions at the derivation tree of Figure 4a for examples). Rule (8) emits a single word from the vocabulary of the training set. Its weight defines a multinomial distribution over all seen words, for every value of field f, given that the field type is categorical (denoted as *cat* in the grammar) or the special *null* field. Rule (9) is identical but for fields whose type is integer. Function $\text{gen}(f.v)$ generates an integer number given the field value, using either of the following six ways (Liang et al., 2009): identical to the field value, rounding up or rounding down to a multiple of 5, rounding off to the closest multiple of 5 and finally adding or subtracting some unexplained noise ε_+ or ε_- respectively. Each noise is modeled as a geometric distribution, the parameters of which are trained given the value $f.v$. The weight is a multinomial over the six integer generation function choices, given the record field f, times $P(f.v\,|\,gen(f.v).mode)$, which is set to the geometric distribution of noise ε_+ and ε_-, or to 1 otherwise.

Naturally, our grammar can yield several derivation trees for a given input string. Notice the difference between Figure 4a and Figure 4b in emitting the phrases "*with temperatures between 10*" and "*and 20 degrees .*". In Figure 4a, the field *min* (whose record is **Temperature**) spans the entire phrase, whereas in Figure 4b the phrase is split in two parts. The *null* field emits "*with temperatures*" and the *min* field emits "*between 10*". Analogously, in the derivation tree in Figure 4a, the field *max* emits the first three words, "*and 20 degrees*", then the *null* emits the full-stop on its own null field of the same record (very common situation in case of punctuation marks). In the derivation tree of Figure 4b, however, the whole phrase is spanned by the field *max*.

3.3 Generation

So far we have defined a probabilistic grammar which captures the structure of a database **d** with records and fields as intermediate non-terminals, and words w (from the associated text) as terminals. The mapping between **d** and w is unknown and thus the intermediate multinomials (see the rule weights of G_{GEN} in Table 1) define a distribution over hidden correspondences h between records, fields and their values. Given an input scenario from a database **d** we can generate its corresponding text using the grammar in Table 1.

On a high-level our generation procedure can be described as follows. We first select the length N of the output text (we defer discussion on how we achieve this to Section 4.3). Then, we apply our grammar to the "empty" document by building derivation trees in a bottom-up fashion, starting from the lexical rules $r \in G_{SURF}$. For each word position in the document we emit a k-best list of candidate words drawn from the corresponding distributions, given the values of the fields of the records in **d**; then, we apply the rest of the rules $r \in G_{CS}$, keeping a list of k-best partial derivations *and* partially generated text in each node[4], until we reach the root symbol S spanning the whole document. Finally, we reconstruct the top-scoring generated string at the root of the tree, by following the pointers of the best derivation, down to the lexical rules that emit the words of the final document. In order to guarantee the grammaticality of the final output text, we rescore the k-best lists at each node by applying external linguistic knowledge, such as n-gram language models and head dependency-style models, on the partially generated substrings.

In analogy to parsing, this procedure amounts to finding the most likely derivation, i.e., sequence of rewrite rules for a given input. Note, that there is a subtle difference between syntactic parsing and generation. In the former case, we observe a string of words and our goal is to find the most probable syntactic structure, i.e., hidden correspondence $\hat{h}$. In generation, however, as described above, the string is not observed; instead, we must thus find the best text $\hat{g}$, by maximizing both over h and g (the latter is achieved with the use of external linguistic knowledge via rescoring), where $g = g_1 \dots g_N$ is a sequence of words licensed by G_{CS} and G_{SURF}. More formally:

$$\hat{g} = f\Big(\arg\max_{g,h} P\big((g,h)\big)\Big) \tag{4}$$

where f is a function that takes as input a derivation tree (g,h) and returns $\hat{g}$. We use a modified version of the CYK parser (Kasami, 1965; Younger, 1967) to find $\hat{g}$. Optimizing over both h and g is intractable, so we approximate f by pruning the search space as we explain in Section 3.5.

In the following, we we will use the framework of deductive proof systems (Shieber, Schabes, & Pereira, 1995) in order to describe our decoder. We first present a basic adaptation of the CYK algorithm to our task and give a concrete decoding procedure that generates text, using a chart data structure (Section 3.4). We then extend the basic decoder into a k-best decoder, by integrating external linguistic knowledge in an attempt to improve the quality of the output. The basic decoder naively only optimizes function f over h, whereas the extended version maximizes both h and g, approximately. Note that the framework of deductive proof systems is used here for convenience. It

4. We use an efficient method that compresses the stored substrings considerably, following the work of Chiang (2007); see equation (12) in Section 3.6.

Items: $[A, i, j]$
$R(A \rightarrow B)$
$R(A \rightarrow BC)$

Axioms: $[W, i, i+1] : s \qquad W \rightarrow g_{i+1},\ g_{i+1} \in \{\alpha, \text{gen}()\}$

Inference rules:

(1) $$\frac{R(A \rightarrow B) : s\ [B, i, j] : s_1}{[A, i, j] : s \cdot s_1}$$

(2) $$\frac{R(A \rightarrow B\ C) : s\ [B, i, k] : s_1\ [C, k, j] : s_2}{[A, i, j] : s \cdot s_1 \cdot s_2}$$

Goal: $[S, 0, N]$

Figure 5: The basic decoder deductive system. Productions $A \rightarrow B$ and $A \rightarrow B\ C$ can be any of the G_{CS} rules in Figure 1; features on grammar non-terminals are omitted for the sake of clarity.

provides a level of abstract generalization for a number of algorithms. Examples include the recogntion of a sentence according to a grammar, learning inside and outside weights, Viterbi search, and in our case generating text (see Goodman, 1999 for more details).

3.4 Basic Decoder

Analogously to a parser, our decoder can be generally defined as a set of weighted **items** (some of which are designated axioms and others are **goals**, i.e., items to be proven) and a set of inference rules of the form:

$$\frac{I_1 : s_1 \ldots I_k : s_k}{I : s} \Phi$$

which can be interpreted as follows: if all items I_i (i.e., the antecedents) have been first proven with weight (or score) s_i, then item I (i.e., the consequent) is provable, with weight s provided the side condition Φ holds. The decoding process begins with the set of axioms, and progressively applies the inference rules, in order to prove more items until it reaches one of the designated goals.

Our basic decoder is specified in Figure 5 and consists of four components, a class of items, a set of axioms, a set of inference rules and a subclass of items, namely the goal items. Following the work of Goodman (1999), items in our system take two forms: $[A, i, j]$ indicates a generated span from i to j, rooted at non-terminal A; $R(A \rightarrow B)$ or $R(A \rightarrow B\ C)$ corresponds to any of the content selection production rules of G_{CS} with one or two non-terminals on the right hand side. Axioms correspond to each individual word generated by the surface realization grammar rules G_{SURF} (see (8) and (9) in Table 1). Our inference rules follow two forms, one for grammar production rules with one non-terminal on the right hand side, and another one for rules with two non-terminals. For example, inference rule (1) in Figure 5 combines two items, namely a rule of the form $A \rightarrow B$ with weight s and a generated span $[B, i, j]$ with weight s_1 rooted at B, and results to a new generated span $[A, i, j]$ with weight $s \cdot s_1$, rooted at A. Finally, our system has one goal, $[S, 0, N]$, where S is the root node of the grammar and N the (predicted) length of the generated text. The time complexity is

$O(n^3)$, as in the case of CYK algorithm. We could have converted our grammar rules in Chomsky normal form (CNF) and implemented the original CYK algorithm. Note that our grammar is not in CNF, since it contains unary productions of the type A $\rightarrow$ B, i.e., with non-terminal symbols on the right-hand side as well. We chose to directly implement inference rules (1) and (2) instead (see Figure 5), since we know that the arity of our grammar is at most 2 and were thus able to avoid a blow-up in the number of derived rules.

Now that we have defined the parsing strategy, we need a way to find the most likely derivation; the pseudocode of Figure 6 gives the generation algorithm for the basic decoder. It uses an array $chart[A,i,j]$, the cells of which get filled with sets of weights of items. It also uses an identical array $bp[A,i,j]$ that stores back-pointers to the antecedents of each item rooted at A, as well as the actual generated words when processing the lexical rules $r \in G_{SURF}$ (abusing somewhat the traditional interpretation of a back-pointer array, as a storage of pointers to antecedent chart items). The size of the chart and the back-pointer array are set to the pre-defined number of N words we want to generate (Section 4.3). The procedure begins by first filling in the 'diagonal' cells of the *chart* with unary spans rooted at W, with the weights of the lexical rules $r \in G_{SURF}$. Equivalently, the back-pointers array takes the corresponding generated word. Note that in a conventionial parsing procedure, we always assume that the 'diagonal' cells of the *chart* are already filled in with the actual words of the underlying sentence. In our case, we only assume a fixed-size *chart* with an empty 'diagonal', which gets filled in with the top scoring words emitted by the lexical rules of our grammar. Next, items are visited and combined in order, i.e., smaller spans come before larger spans. Given the way our grammar is constructed, items rooted in F (corresponding to fields) will come before items rooted in R (records) and ultimately before S. At any particular point in the *chart*, the algorithm considers all the antecedent items that can be proven given the rules of G_{CS} and stores the highest scoring combination. Finally, we can construct the resulting string $\hat{g}$ by recursively visiting $bp[S,0,N]$. We trace the back-pointers of each item to its antecedents down to the words g_i emitted by the axioms.

3.5 *k*-best Decoding

The basic decoder described so far will produce the best derivation tree of the input **d** given the grammar G_{GEN} which unfortunately may not correspond to the best generated text. In fact, the output will often be poor as the model has no notion of what constitutes fluent language. The grammar encodes little knowledge with regard to syntactic well-formedness and grammatical coherence. Essentially, surface realization boils down to the word bigram rules (6) and (7) and the lexical rules in G_{SURF}. The word bigram rules inject some knowledge about word combinations into the model, but this kind of information is usually sparse and cannot capture longer range dependencies.

The generation process in Figure 6 picks the top scoring words emitted by the lexical production rules (lines 3–5), in order to produce the best derivation at the root node S. Instead, it would be preferable if we added to the *chart* a list of the top k words (as well as a list of the top k items $[B,i,j]$, $[C,j,k]$ for each production rule $r \in G_{CS}$), and thus produced a k-best list of derivations (with their associated strings) at the root node. This can be done efficiently using the lazy algorithm found in the work of Huang and Chiang (2005). Then, once the generation process is finished, we can use a language model such as higher order n-grams, or head dependency-style rules to rescore the k-best lists of generated strings directly (see also Charniak & Johnson, 2005 and Liang, Bouchard-Côté, Klein, & Taskar, 2006 for application of a similar idea to parsing and machine translation,

```
function DECODE(G_GEN, d, N)
    for i ← 0...N do
        for all r : W → g_{i+1} ∈ G_SURF do
            chart[W, i, i+1] ← [W, i, i+1] : s
            bp[W, i, i+1] ← g_{i+1}                                  ▷ store actual word g_{i+1}
        end for
    end for
    for l ← 2...N do
        for all i, k, j so that j − i = l and i < k < j do
            for all items [B, i, j] or [B, i, k], [C, k, j] inferable from chart and rules r ∈ G_CS do
                if r is of the form A → B then
                    chart[A, i, j] ← max([B, i, j] : s_1 × P(r))
                    bp[A, i, j] ← argmax([B, i, j] : s_1 × P(r))
                end if
                if r is of the form A → B C then
                    chart[A, i, j] ← max(chart[B, i, k] × chart[C, k, j] × P(r))
                    bp[A, i, j] ← argmax([B, i, k] : s_1 × [C, k, j] : s_2 × P(r))
                end if
            end for
        end for
    end for
    return chart[S, 0, N], bp[S, 0, N]
end function
```

Figure 6: Generation procedure for the basic decoder.

respectively). Although this method is fast, i.e., linear in k, we would practically have to set k very high and search among exponentially many possible generations for a given input.

A better solution, which is common practice in machine translation, is to rescore the derivation trees online. Chiang (2007) intersects a PCFG grammar with a weighted finite state automaton (FSA), which represents a n-gram language model; the states of the FSA correspond to $n-1$ terminal symbols. The resulting grammar is also a PCFG that incorporates the FSA. Similarly, we can intersect our grammar with an ensemble of external probabilistic models, provided that they express a regular language. The most probable generation $\hat{g}$ is then calculated as:

$$\hat{g} = f\Big(\arg\max_{g,h} p(g) \cdot p(g,h\,|\,\mathbf{d})\Big) \tag{5}$$

where $p(g,h\,|\,\mathbf{d})$ is the decoding likelihood for a sequence of words $g = g_1 \dots g_N$ of length N and the hidden correspondence h that emits it, i.e., the likelihood of our grammar for a given database input scenario $\mathbf{d}$. $p(g)$ is a measure of the quality of each output and could for instance be provided by a language model (see Section 4.2 for details on how we estimate $p(g,h\,|\,\mathbf{d})$ and $p(g)$). In theory, the function f above should optimize h and g jointly, thus admitting no search errors. In practice, however, the resulting grammar after the intersection is prohibitively large, and calls for pruning of the search space. In the following we show how to extend the basic generation decoder in Figure 5 by intersecting it (linearly) with an ensemble of external probabilistic models.

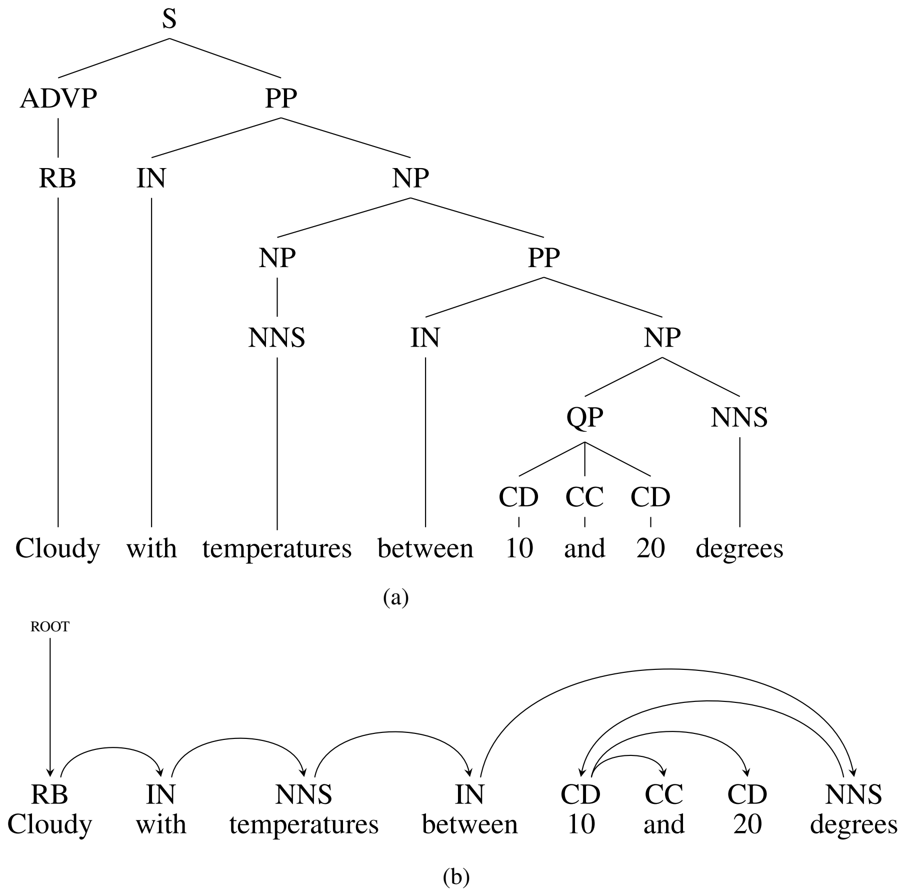


Figure 7: Phrase structure tree and dependency graph for the same sentence.

In addition to *n*-gram language models which are routinely used as a means of ensuring lexical fluency and some rudimentary grammaticality, we also inject syntactic knowledge into our generator. We represent syntactic information in the form of directed dependencies which could potentially capture long range relationships beyond the horizon of a language model. Figure 7 shows a dependency-style representation for the sentence "*Cloudy with temperatures between 10 and 20 degrees*" and its corresponding phrase structure. The dependency graph in Figure 7b captures grammatical relations between words via directed edges from syntactic heads to their dependents (e.g., from a verb to its subject or from a noun to a modifying adjective). Edges can be labeled to indicate the type of head-dependent relationship (e.g., subject or object) or unlabeled as shown in the figure. Formally, a dependency structure D is a set of dependency pairs $\langle w_h, w_a \rangle$ of a head w_h and an argument word w_a, respectively. In general, the argument is the modifier, object or complement; the head most of the times determines the behavior of the pair. In Figure 7b, *cloudy* is the head of *with*, *with* is the head of *temperature*, and so on. $D(w_h)$ returns a set of dependency pairs whose head is w_h, e.g., $D(\mathit{10}) = \{\mathit{and, 20}\}$.

Previous work (Ratnaparkhi, 2002) has incorporated dependency information into surface realization more directly by generating a syntactic dependency tree rather than a word sequence. The underlying probabilistic model predicts each word by conditioning on syntactically related words (i.e., parent, grandparent, and siblings). Importantly, this approach requires a corpus that has been annotated with dependency tree structures. We obviate the need for manual annotation by considering dependency structures that have been induced automatically in an unsupervised fashion. For this, we use the Dependency Model with Valence (DMV; Klein & Manning, 2004), however, there is nothing inherent in our formulation that restricts us to this model. Any other unsupervised model that learns dependency structures in a broadly similar fashion (e.g., captures the attachment likelihood of an argument to its head) could have been used instead with the proviso that it operates on structures that are isomorphic to the derivation trees generated by our grammar. This is necessary if the intersecting dependency model expresses (up to) a context-free language, since we formulate our model also as a CFG[5].

Finally, note that although we work with two external information sources (i.e., language models and dependencies), the framework we propose applies to an arbitrary number of models expressing a regular language. For instance, we could incorporate models that capture dependencies relating to content selection such as field n-grams, however we leave this to future work.

3.6 Extended Decoder

We begin by introducing some notation. We define two functions p and q which operate over M surface-level models and strings $a = a_1 \dots a_l$, of length l, with $a_i \in V \cup \{\star\}$. V is the vocabulary of the observed text w (obtained from the training corpus), and the $\star$ symbol represents the elided part of a string. Recall that our k-best decoder needs to keep a list of generated sub-strings a at each node, for rescoring purposes. Note that these sub-strings are (potentially) different from the observed text w; the top-scoring string on the root node essentially collapses to the final generated text g. Storing lists of whole sub-strings generated so far at each node, would require considerable amounts of memory. To avoid this we define a function $q(a)$ that stores the essential minimum string information needed for each of the surface-level models (the $\star$ symbol stands for the omitted parts of a string) at each step, in order to correctly compute the rescoring weight. Function $p(a)$ essentially calculates the rescoring weight for a given string, by linearly interpolating the scores of each individual model m_i with a weight β_i. Therefore applying $p(a)$ in a bottom-up fashion (see the extended decoder of Figure 8) on the output of $q(a)$ allows us to correctly compute the rescoring weight of each model for the whole document incrementally. More formally:

$$p(a) = \sum_{i}^{M} \beta_i p_{m_i}(a) \qquad \text{s.t.} \sum_{i}^{M} \beta_i = 1 \tag{6}$$

In our setting, we make use of a language model (p_{m_1}) and a dependency model (p_{m_2}):

$$p_{m_1}(a_1 \dots a_l) = \prod_{\substack{n \le i \le l \\ \star \notin \{a_{i-n+1},\dots,a_i\}}} P_{LM}(a_i | a_{i-n+1} \dots a_{i-1}) \tag{7}$$

$$p_{m_2}(a_1 \dots a_l) = P_{DEP}\big(D(a_h)\big), \text{ where } a_h \in \{a_1, \dots, a_l\} \tag{8}$$

The function p_{m_1} computes the LM probabilities for all complete n-grams in a string; P_{LM} returns the probability of observing a word given the previous $n-1$ words. p_{m_2} returns the probability of the

5. Intersecting two CFGs is undecidable, or PSPACE-complete if one CFG is finite (Nederhof & Satta, 2004).

$a_1 \dots a_l$	$p_{m_1}(a_1 \dots a_l)$	$q_{m_1}(a_1 \dots a_l)$
mostly cloudy ,	P_{LM}(,|mostly cloudy)	mostly cloudy $\star$ cloudy ,
with a	1	with a
mostly cloudy $\star$ cloudy , with a	P_{LM}(with|cloudy ,) $\times$ P_{LM}(a|, with)	mostly cloudy $\star$ with a

Table 2: Example values for functions p_{m_1} and q_{m_1} for the phrase *"mostly cloudy, with a"*. We assume a 3-gram language model.

dependency model on the dependency structure D headed by word a_h. For a dependency structure D, each word a_h has dependants $deps_D(a_h, left)$ that attach on its left and dependents $deps_D(a_h, right)$ that attach on its right. Equation (9) recursively defines the probability of the dependency $D(a_h)$ rooted at a_h (Klein & Manning, 2004):

$$\begin{aligned} P_{DEP}\big(D(a_h)\big) = \prod_{dir \in [left, right]} \Big[& \prod_{deps_D(a_h, dir)} P_{\text{STOP}}(\neg\text{STOP}|a_h, dir, adj) \\ & P_{\text{CHOOSE}}(a_a|a_h, dir) P_{DEP}\big(D(a_a)\big)\Big] \\ & P_{\text{STOP}}(\text{STOP}|a_h, dir, adj) \end{aligned} \tag{9}$$

P_{STOP} is a binary multinomial indicating whether to stop attaching arguments to a head word a_h given their direction, i.e., left or right, and their adjacency, i.e., whether they are directly adjacent to a_h or not. P_{CHOOSE} is a multinomial over all possible argument words given a_h and the direction of attachment. We next define function $q(a)$ which returns a set of M strings, one for each model m_i (we will use it shortly to expand the lexical items $[A, i, j]$ of the basic decoder in Figure 5).

$$q(a) = \langle q_{m_1}(a), \dots, q_{m_M}(a) \rangle \tag{10}$$

$$\tag{11}$$

$$q_{m_1}(a_1 \dots a_l) = \begin{cases} a_1 \dots a_{n-1} \star a_{l-n+2} \dots a_l & \text{if } l \geq n \\ a_1 \dots a_l & \text{otherwise} \end{cases} \tag{12}$$

$$\tag{13}$$

$$\underset{1 \leq k \leq l}{\mathrm{q_{m_2}}}(a_1 \dots a_k a_{k+1} \dots a_l) = \begin{cases} a_l & \text{if } l = 1 \\ q_{m_2}(a_1 \dots a_k) & \text{if } p_{m_2}(a_1 \dots a_k) \geq \\ & \quad p_{m_2}(a_{k+1} \dots a_l) \\ q_{m_2}(a_{k+1} \dots a_l) & \text{otherwise} \end{cases} \tag{14}$$

Function $q_{m_1}(a)$ compresses the string a, by eliding words when all their n-grams have been recognized. We thus avoid storing the whole sub-generation string, produced by the decoder so far, as mentioned earlier. Table 2 gives example values for $p_{m_1}(a)$ and $q_{m_1}(a)$ for the phrase *"mostly cloudy, with a*. Function $q_{m_2}(a)$ returns the head of the string a. As we progressively combine substrings $(a_1 \dots a_k)$ and $(a_{k+1} \dots a_l)$ together, for any $1 \leq k \leq l$, and their head words $a_{h_1} \in \{a_1, \dots, a_k\}$ and $a_{h_2} \in \{a_{k+1}, \dots, a_l\}$, function $q_{m_2}(a)$ returns either a_{h_1} or a_{h_2}. The probability P_{DEP} decides whether a_{h_1} attaches to a_{h_2} or vice versa, thus augmenting $D(a_{h_1})$ with the pair $\langle a_{h_1}, a_{h_2} \rangle$ or $D(a_{h_2})$ with $\langle a_{h_2}, a_{h_1} \rangle$, respectively.

Items: $[A,i,j;q(g_i^j)]$
$R(A \to B)$
$R(A \to BC)$

Axioms: $[W,i,i+1;q(g_i^{i+1})] : s \cdot p(g_i^{i+1}) \qquad W \to g_{i+1},\ g_{i+1} \in \{\alpha, \text{gen}()\}$

Inference rules:

$$(1) \quad \frac{R(A \to B) : s \ [B,i,j;q(g_i^j)] : s_1}{[A,i,j;q(g_i^j)] : s \cdot s_1 \cdot p(g_i^j)}$$

$$(2) \quad \frac{R(A \to B\ C) : s \ [B,i,k;q(g_i^k)] : s_1 \ [C,k,j;q(g_k^j)] : s_2}{[A,i,j;q(g_i^j)] : s \cdot s_1 \cdot s_2 \cdot p(g_i^j)}$$

Goal: $[S,0,N;q(\langle s\rangle^{n-1} g_0^N \langle /s\rangle)]$

Figure 8: Extended decoder using the rescoring function $p(g)$. Productions $A \to B$ and $A \to B\ C$ can be any of the G_{CS} rules in Figure 1; features on grammar non-terminals are omitted for the sake of clarity.

Note that equation (14) evaluates whether every word should attach to the left or right of every other head word, and therefore essentially collapses to:

$$P_{m_{dep}} = P_{DEP}\big(D(a_h)\big) = P_{\text{STOP}}(\neg\text{STOP}|a_h, dir, adj) P_{\text{CHOOSE}}(a_a|a_h, dir) \\ P_{\text{STOP}}(\text{STOP}|a_h, dir, adj) \tag{15}$$

For example, in the case of $p_{m_2}(a_1 \dots a_k)$, a_h becomes one of $a_1 \dots a_k$, a_a is one of $a_{k+1} \dots a_l$, $dir = right$ and adj is true if $a_h = a_k$ and $a_a = a_{k+1}$.

We are now ready to extend the basic decoder in Figure 5, so that it includes the rescoring function $p(g_i^j)$ over a generated sub-string $g_i \dots g_j$. The new deduction system is specified in Figure 8. Items $[A,i,j]$ become now $[A,i,j;q(g_i^j)]$; they represent derivations from g_i to g_j rooted at the non-terminal A and augmented with model-specific strings as defined above; in other words, they include the compressed sub-generations with elided parts and their head word. Analogously, our goal item now includes $q\left(\langle s\rangle^{n-1} g_0^N \langle /s\rangle\right)$. Note that g_0^N is augmented with $(n-1)$ start symbols $\langle s\rangle$ and an end symbol $\langle /s\rangle$. This is necessary for correctly computing n-gram probabilities at the beginning and end of the sentence. Figure 9 shows example instantiations of the inference rules of our extended decoder.

The generation procedure is identical to the procedure described for the basic decoder in Figure 6, save the exponential more items that need to be deducted. Recall that the *chart* in Figure 6 stores at each cell $chart[A,i,j]$ the set of combined weights of cells that correspond to the proved antecedents of item $[A,i,j]$. The new $chart'$ for the extended decoder equivalently stores a set of lists of weights at each cell position $chart'[A,i,j]$. The list contains the items $[A,i,j;q(g_i^j)]$ that have the same root non-terminal A and span between i and j, but a different set $q(g_i^j)$, sorted best-first. The running time of integrating the LM and DMV models is $O(N^3|V|^{4(n-1)}|P|)$, where V is the output vocabulary and P the vocabulary used in the DMV. When using a lexicalized dependency model,

$$\frac{R\left(\mathrm{R}(\mathrm{skyCover}_1.t) \rightarrow \mathrm{FS}(\mathrm{temp}_1, start)\ \mathrm{R}(\mathrm{temp}_1.t)\right) : s \quad [\mathrm{FS}(\mathrm{temp}_1, start), 1, 2; \langle \text{with, IN} \rangle] : s_1 \ [\mathrm{R}(\mathrm{temp}_1.t), 2, 8; \langle \text{a low} \star \text{15 degrees, JJ} \rangle] : s_2}{[\mathrm{R}(\mathrm{skyCover}_1.t), 1, 8; \langle \text{with a} \star \text{15 degrees, JJ} \rangle] : s \cdot s_1 \cdot s_2 \cdot p(\langle \text{with a} \star \text{15 degrees, JJ} \rangle)}$$

$$\frac{R\left(\mathrm{FS}(\mathrm{windSpeed}_1, min) \rightarrow \mathrm{F}(\mathrm{windSpeed}_1, max)\ \mathrm{FS}(\mathrm{windSpeed}_1, max)\right) : s \quad [\mathrm{F}(\mathrm{windSpeed}_1, max), 3, 4; \langle \text{high, JJ} \rangle] : s_1 \ [\mathrm{FS}(\mathrm{windSpeed}_1, max), 4, 5; \langle \text{15, CD} \rangle] : s_2}{[\mathrm{FS}(\mathrm{windSpeed}_1, min), 3, 5; \langle \text{high 15, JJ} \rangle] : s \cdot s_1 \cdot s_2}$$

$$\frac{R\left(\mathrm{F}(\mathrm{windDir}_1, mode) \rightarrow \mathrm{W}(\mathrm{windDir}_1, mode)\right) : s \ [\mathrm{W}(\mathrm{windDir}_1, mode), 3, 4; \langle \text{southeast, JJ} \rangle] : s_1}{[\mathrm{F}(\mathrm{windDir}_1, mode), 3, 4; \langle \text{southeast, JJ} \rangle] : s \cdot s_1}$$

Figure 9: Inference rules in the extended decoder for productions (2), (4), and (7) from Table 1 (WEATHERGOV domain). The strings in $\langle \ldots \rangle$, correspond to the output of the functions $q_{m_{lm}}$ and $q_{m_{dep}}$. We adopt an unlexicalized dependency model, trained on POS tags derived from the Penn Treebank project (Marcus et al., 1993). In the first example IN corresponds to the word *with* and JJ to the word *low*, in the second example JJ corresponds to the word *high* and CD to the number *15*, whereas in the third example JJ corresponds to the word *southeast*.

P collapses to V, otherwise it contains the part-of-speech (POS) tags for every $g_i \in V$. Notice that rule (2) in Figure 8 combines two items that contain at most $2(n-1)$ words, hence the exponent $4(n-2)$. This running time is too slow to use in practice, so as we explain below we must adopt some form of pruning in order to be able to explore the search space efficiently.

3.7 Approximate Search

Consider the task of deriving a k-best list of items $\mathbf{L}([A,i,j;q(g_i^j)])$ for the deducted item $[A,i,j;q(g_i^j)]$ of rule (2) in the extended decoder of Figure 8. An item $L_m([A,i,j;q(g_i^j)])$ at position m of the list, with $1 \leq m \leq k$, takes the form $[A,i,j;q(g_m|_i^j)]$. An example of this procedure is shown in Figure 10. The grid depicts all possible combinations of items $[B,i,k;q(g_i^k)]$ and $[C,k,j;q(g_k^j)]$ as inferred by a rule of the form $R(A \rightarrow B\ C)$ with their corresponding weights. Any of the k^2 combinations can be used to create the resulting k-best list shown at the bottom of the figure, and store it on the cell of $chart'[A,i,j]$. However, we only want to keep k items, so most of them are going to be pruned away. In fact, the grid of the example can be in the worst case a *cube*, i.e., can hold up to two three dimensions, one for all the rules $A \rightarrow B\ C$ with the same left hand-side non-terminal A, and two for the corresponding items rooted on B and C[6]; this calls for the calculation of k^3 combinations. A better approach is to apply *cube pruning* (Chiang, 2007; Huang & Chiang, 2005), i.e., to compute only a small corner of the grid and prune items out on the fly, thus obviating the costly computation of all k^3 combinations.

6. The deducted item $[R(\mathrm{skyCover}_1.t);q(g_1^8)]$ of Figure 10 can also be inferred by the rule $R(R(\mathrm{skyCover}_1.t) \rightarrow R(\mathrm{windSpeed}_1.t)\ FS(\mathrm{windSpeed}_1, start))$ (and its corresponding antecedent items) or the rule $R(R(\mathrm{skyCover}_1.t) \rightarrow R(\mathrm{rainChance}_1.t)\ FS(\mathrm{rainChance}_1, start))$, and so on. We illustrate only a slice of the cube, depicting the enumeration of k-best lists for a fixed grammar rule, for the sake of clarity.

		$[\mathrm{FS}(\mathrm{temp}_1, \mathit{start}), 1, 2; \langle \text{with, IN} \rangle]$	$[\mathrm{FS}(\mathrm{temp}_1, \mathit{start}), 1, 2; \langle \text{a, DT} \rangle]$	$[\mathrm{FS}(\mathrm{temp}_1, \mathit{start}), 1, 2; \langle \text{around, RB} \rangle]$
		.95	.93	.91
$[\mathrm{R}(\mathrm{temp}_1.t), 2, 8; \langle \text{a low} \star \text{15 degrees, JJ} \rangle]$	.56	.40	.25	.20
$[\mathrm{R}(\mathrm{temp}_1.t), 2, 8; \langle \text{low around} \star \text{15 degrees, JJ} \rangle]$	.54	.35	.30	.17
$[\mathrm{R}(\mathrm{temp}_1.t), 2, 8; \langle \text{a low} \star \text{around 17, RB} \rangle]$	.44	.15	.08	.10

$$\Downarrow$$

$$\left(\begin{array}{ll}
[\mathrm{R}(\mathit{skyCover}_1.t), 1, 8; \langle \text{with a} \star \text{15 degrees, JJ} \rangle & : .40 \\
[\mathrm{R}(\mathrm{skyCover}_1.t), 1, 8; \langle \text{with low} \star \text{15 degrees, JJ} \rangle] & : .35 \\
[\mathrm{R}(\mathrm{skyCover}_1.t), 1, 8; \langle \text{a a} \star \text{15 degrees, JJ} \rangle] & : .25 \\
[\mathrm{R}(\mathrm{skyCover}_1.t), 1, 8; \langle \text{around low} \star \text{15 degrees, RB} \rangle] & : .17 \\
[\mathrm{R}(\mathrm{skyCover}_1.t), 1, 8; \langle \text{with a} \star \text{around 17, RB} \rangle] & : .15 \\
\cdots &
\end{array} \right)$$

Figure 10: Computing an exhaustive list for the deducted item $[R(\mathrm{skyCover}_1.t); q(g_1^8)]$ via application of inference rule (2) of the extended decoder in Figure 9. The antecedent items are the rule $R(R(\mathrm{skyCover}_1.t) \rightarrow R(\mathrm{temp}_1.t)\ FS(\mathrm{temp}_1, \mathit{start}))$ and the items $[R(\mathrm{temp}_1.t), 2, 8; q(g_2^8)]$, $FS(\mathrm{temp}_1, \mathit{start}), 1, 2; q(g_1^2)]$. The figure shows a slice of the cube, for the particular rule; on each side of the grid are the lists of the top three candidate items for each antecedent item, sorted best-first. Numbers in the grid represent the total score for each combination.

Consider Figure 11 as an example. Each side of the grid shows the lists of the top three items for each antecedent item. Numbers on the grid represent the total score for each combination. Figures 11b–11d illustrate the enumeration of the top three combinations in best-first order. Cells in gray represent the frontiers at each iteration; cells in black are the resulting top three items. The basic intuition behind cube pruning is that for a pair of antecedent items $u_1 = [B, i, k; q(g_i^k)]$, $u_2 = [C, k, j; q(g_k^j)]$ and their sorted k-best lists $\mathbf{L}(u_1)$, $\mathbf{L}(u_2)$, the best combinations should lie close to the upper-left corner of the grid. In the example, the 3-best list of the nodes $u_1 = [R(\mathrm{temp}_1.t), 2, 8; q(g_2^8)]$

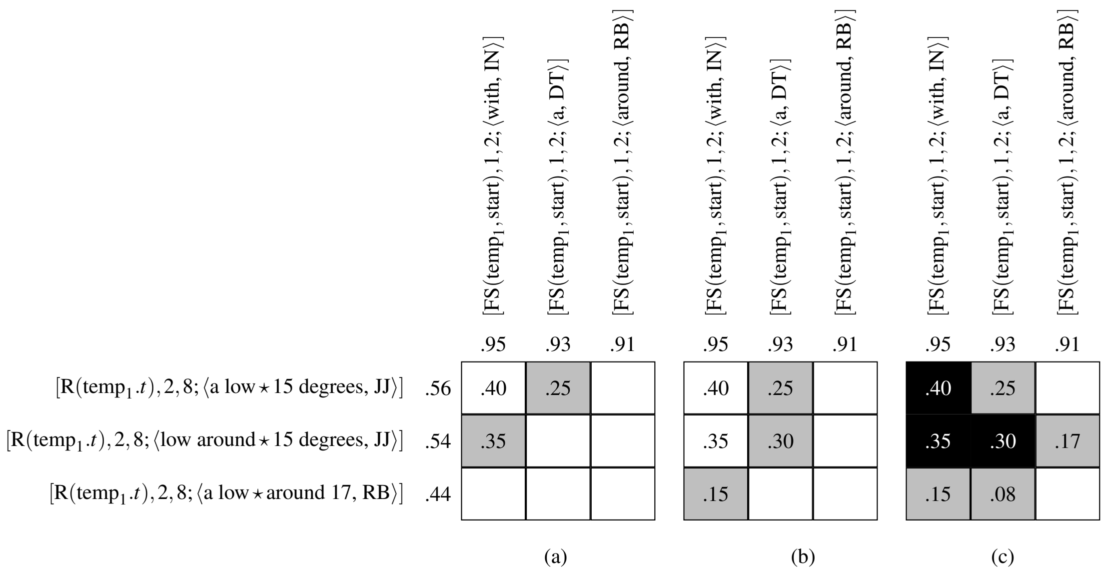


Figure 11: Computing item combinations for $u_1 = [R(\text{temp}_1.t), 2, 8; q(g_2^8)]$ and $u_2 = [FS(\text{temp}_1, start), 1, 2; q(g_1^2)]$ using cube pruning. In (a)–(c) we enumerate the combinations of items in order to construct a resulting k-best list as described in the text.

and $u_2 = [FS(\text{temp}_1, start), 1, 2; q(g_1^2)]$ are:

$$\mathbf{L}(u_1) = \Big[\langle \text{a low} \star \text{15 degrees, JJ}\rangle, \langle \text{low around} \star \text{15 degrees, JJ}\rangle, \langle \text{a low} \star \text{around 17, RB}\rangle\Big]$$
$$\mathbf{L}(u_2) = \Big[\langle \text{with, IN}\rangle, \langle \text{a, DT}\rangle, \langle \text{around, RB}\rangle\Big]$$

and intuitively the best combination should be the derivation on the top left corner[7]:

$$\Big(L_1(u_1), L_1(u_2)\Big) = \Big(\langle \text{a low} \star \text{15 degrees, JJ}\rangle, \langle \text{with, IN}\rangle\Big) = \langle \text{with a} \star \text{15 degrees, IN}\rangle$$

In cases where the combination cost, i.e., the score of the grammar rule multiplied with the rescoring weight $p(g)$, is negligible, we could start enumerating item combinations in the order shown in Figures 11b–11c, starting from $(L_1(u_1), L_1(u_2))$ and stopping at k. Since the two lists are sorted it is guaranteed that $L_2(u_1)$, i.e., the second item in the k-best list of u_1 is either $(L_1(u_1), L_2(u_2))$ or $(L_2(u_1), L_1(u_2))$ (in the example of Figure 11b it is the latter). We thus select it and move on to compute its neighboring combinations, and so on.[8] For the computation of the k-best lists of the axioms $[W, i, i+1; q(g_i^{i+1})]$, we enumerate the top-$k$ terminal symbols g_{i+1}.

If we take into account the combination cost, the grid is non-monotonic, and therefore the best-first guarantee no longer holds as we enumerate neighbors in the fashion just described. Huang and Chiang (2007) argue that the loss incurred by the search error is insignificant compared to the speedup gained. In any case, to overcome this, we compute the resulting k-best list, by first adding

7. Note that the head of the sub-generation fragment has shifted to the head of L_2.
8. Contrary to Huang and Chiang (2007) we use probabilities instead of log scores in the computation of the item combinations, hence we select the biggest scoring combinations.

the computed item combinations in a temporary buffer, and then resort it after we have enumerated a total of k combinations.

3.8 Learning

We represent our grammar and each input scenario as a weighted hypergraph (Gallo, Longo, Pallottino, & Nguyen, 1993). We follow the procedure proposed by Klein and Manning (2001) which allows to transform any CFG to a hypergraph. In order to learn the weights of the grammar rules we directly estimate them on the hypergraph representation using the EM algorithm. Formally, the objective we are trying to optimize factorizes into:

$$P(\mathbf{r}, r_i.\mathbf{f}, w|\mathbf{d}) = \prod_i P(r_i|\mathbf{d}) \prod_j P(r_i.f_j|r_i) \prod_k P(w_j|r_i.f_j, r_i) \tag{16}$$

where $\mathbf{r}$ is the set of all record tokens r_i. Given a training set of scenarios with database records $\mathbf{d}$ and *observed* text w we maximize the marginal likelihood of the data, while summing out record tokens r_i and their fields $r_i.\mathbf{f}$, which can be regarded as latent variables:

$$\arg\max_{\theta} \prod_{(w,\mathbf{d})} \sum_{\mathbf{r}, r_i.\mathbf{f}} p(\mathbf{r}, r_i.\mathbf{f}, w|\mathbf{d}; \theta), \tag{17}$$

where θ are the multinomial distributions or weights of G_{GEN}. The EM algorithm alternates between the E-step and the M-step. In the E-step we compute the expected counts for the rules using a dynamic program similar to the inside-outside algorithm (Li & Eisner, 2009). Then in the M-step, we optimise θ by normalising the counts computed in the E-step. We initialise EM with a uniform distribution for each multinomial distribution and applied add-0.001 smoothing to each multinomial in the M-step. On average, EM converged for all datasets after 15 iterations. Note that the n-gram language model and the dependency model are trained externally, hence their parameters are not optimized alongside our model. The generation procedure for the extended decoder in Figure 8 is implemented using dynamic programming. The choice of the hypergraph representation is merely one of several alternatives. For example, we could have adopted a representation based on weighted finite state transducers (de Gispert, Iglesias, Blackwood, Banga, & Byrne, 2010) since our model describes a regular language both in terms of the PCFG and the surface level models we intersect it with. It is also possible to represent our grammar as a pushdown automaton (Iglesias, Allauzen, Byrne, de Gispert, & Riley, 2011) and intersect it with finite automata representing a language model and dependency-related information, respectively. The choice of the hypergraph representation was motivated by its compactness[9] and the fact that it allows for future extensions of our PCFG with rules which capture more global aspects of the generation problem (e.g., document planning) and which unavoidably result in context-free languages.

4. Experimental Design

In this section we present our experimental setup for assessing the performance of our model. We give details on the datasets we used, explain how our own model was trained, describe the models used for comparison with our approach, and discuss how system output was evaluated.

9. Hypergraphs are commonly used in the machine translation literature to allow for compact encoding of SCFGs even though in some cases they also describe regular languages. For example, this is true for the SCFGs employed in hierarchical phrase-based SMT (Chiang, 2007) which assume a finite input language and do not permit infinite recursions.

4.1 Data

We used our system to generate soccer commentaries, weather forecasts, and spontaneous utterances relevant to the air travel domain (examples are given in Figure 1). For the first domain we used the dataset described in the work of Chen and Mooney (2008), which consists of 1,539 scenarios from the 2001–2004 Robocup game finals (henceforth ROBOCUP). Each scenario contains on average $|\mathbf{d}| = 2.4$ records, each paired with a short sentence (5.7 words). This domain has a small vocabulary (214 words) and simple syntax (e.g., a transitive verb with its subject and object). Records in this dataset were aligned manually to their corresponding sentences (Chen & Mooney, 2008). Given the relatively small size of this dataset, we performed cross-validation following previous work (Chen & Mooney, 2008; Angeli et al., 2010). We trained our system on three ROBOCUP games and tested on the fourth, averaging over the four train/test splits.

For weather forecast generation, we used the dataset presented in the work of Liang et al. (2009), which consists of 29,528 weather scenarios for 3,753 major US cities (collected over four days). The vocabulary in this domain (henceforth WEATHERGOV) is comparable to ROBOCUP (345 words), however, the texts are longer ($N = 29.3$) and more varied. On average, each forecast has 4 sentences and the content selection problem is more challenging; only 5.8 out of the 36 records per scenario are mentioned in the text which roughly corresponds to 1.4 records per sentence. We used 25,000 scenarios from WEATHERGOV for training, 1,000 scenarios for development and 3,528 scenarios for testing. This is the same partition used in the work of Angeli et al. (2010).

For the air travel domain we used the ATIS dataset (Dahl, Bates, Brown, Fisher, Hunicke-Smith, Pallett, Pao, Rudnicky, & Shriberg, 1994), consisting of 5,426 scenarios. These are transcriptions of spontaneous utterances of users interacting with a hypothetical online flight booking system. We used the dataset introduced in the work of Zettlemoyer and Collins (2007)[10] and automatically converted their lambda-calculus expressions to attribute-value pairs following the conventions adopted in the study of Liang et al. (2009).[11] Figure 1c shows the output of our conversion process from the original lambda expression $\lambda x.flight(x) \wedge from(x, denver) \wedge to(x, boston) \wedge day\_number\_departure(x, 9) \wedge month\_departure(x, august) \wedge < (arrival\_time(x),\ 16{:}00)$. Given such an expression, we first create a record for each variable (e.g., x). We then assign record types according to the corresponding class types (e.g., variable x has class type *flight*). Next, fields and values are added from predicates with two arguments with the class type of the first argument matching that of the record type. The name of the predicate denotes the field, and the second argument denotes the value (e.g., $from(x, denver)$ is used to fill the record of type **Flight**, since the type of the first argument is also $flight$). The name of the function becomes the field name, (i.e., *from*) and the second argument is set as its value, (i.e., `denver`). Note that some functions have names such as *month_departure*, *month_arrival*, *day_number_arrival*, *day_number_departure* and so on. In order to reduce the resulting number of record types, we created aggregate record types which embed the common information (i.e., departure, or arrival) to a special field. In the example, the function *day_number_departure* is split into the value `departure` of the field *dep/ar* for the record **Day**, and into the field *number* with value `9`. We also defined special record types, such as **Condition** and **Search**. The latter is introduced for every lambda operator and assigned the categorical field *what* with value `flight` which refers to the record type of variable x.

10. The original corpus contains user utterances of single dialogue turns which would result in trivial scenarios. Zettlemoyer and Collins (2007) concatenate all user utterances referring to the same dialogue act, (e.g., book a flight), thus yielding more complex scenarios with longer sentences.
11. See Konstas (2013) for the resulting dataset.

In contrast to the two previous datasets, ATIS has a much richer vocabulary (927 words); each scenario corresponds to a single sentence (average length is 11.2 words) with 2.65 out of 19 record types mentioned on average. Note that the original lambda expressions were created based on the utterance, and thus contain all the necessary information conveyed in the meaning of the text. As a result, all of the converted records in each scenario are mentioned in the corresponding text. Following the work of Zettlemoyer and Collins (2007), we trained on 4,962 scenarios and tested on ATIS NOV93 which contains 448 examples.

4.2 Model Training

Generation in our model amounts to finding the best derivation $(\hat{g}, h)$ that maximizes the product of two likelihoods, namely $p(g,h|\mathbf{d})$ and $p(g)$ (see equation (5)). $p(g,h|\mathbf{d})$ corresponds to the rules of G_{GEN} that generate the word sequence g, whereas $p(g)$ is the likelihood of g independently of $\mathbf{d}$. We estimate $p(g,h|\mathbf{d})$ as described in Section 3.8. Examples of the top scoring items of the multinomial distributions for some of the grammar rules of G_{GEN} are given in Table 3. We obtain an estimate for $p(g)$ by linearly interpolating the score of a language model and DMV (Klein & Manning, 2004).

Specifically, our language models were trained with the SRI toolkit (Stolcke, 2002) using add-1 smoothing.[12] For the ROBOCUP domain, we used a bigram language model given that the average text length is relatively small. For WEATHERGOV and ATIS, we used a trigram language model. We obtained an unlexicalized version of the DMV[13] for each of our domains. All datasets were tagged automatically using the Stanford POS tagger (Toutanova, Klein, Manning, & Singer, 2003) and words were augmented with their part of speech, e.g., *low* becomes *low/JJ*, *around* becomes *around/RB* and so on; words with several parts of speech were duplicated as many times as the number of different POS tags assigned to them by the tagger. For example, the *gust* may act both as a noun and a verb, given their context, hence we keep both augmented forms, i.e., *gust/NNS* and *gust/VBS*. We initialized EM to uniform distributions where a small amount of noise[14] was added over all multinomials (i.e., P_{STOP} and P_{CHOOSE}) to break initial symmetry. Klein and Manning (2004) use a harmonic distribution instead, where the probability of one word heading another is higher if they appear closer to one another. Preliminary results on the development set showed that the former initialization scheme was more robust across datasets.

Our model has two hyperparameters: the number of k-best derivations considered by the decoder and the vector β of weights for model integration. Given that we only interpolate two models whose weights should sum to one, we only need to modulate a single interpolation parameter $0 \leq \beta_{LM} \leq 1$. When β_{LM} is 0, the decoder is only influenced by the DMV and conversely when β_{LM} is 1 the decoder is only influenced by the language model. In the general case, we could learn the interpolation parameters using minimum error rate training (Och, 2003), however this was not necessary in our experiments. We performed a grid search over k and β_{LM} on held-out data taken

12. Adopting a more complex smoothing technique such as Good-Turing (Good, 1953) is usually not applicable in so small vocabularies. The statistics for computing the so called count-of-counts, i.e., the number words occurring once, twice and so on, are not sufficient and lead to poor smoothing estimates.
13. When trained on the WSJ-10 corpus, our implementation of the DMV obtained the same accuracy as reported in the work of Klein and Manning (2004). WSJ-10 consists of 7,422 sentences with at most 10 words after removing punctuation.
14. Repeated runs with different random noise on the WSJ-10 corpus yielded the same results; accuracy stabilized around the 60th iteration (out of 100).

Weight Distribution	Top-5 scoring items
$P(\alpha \mid$ **pass**, *from*, `purple2`$)$	purple2, a, makes, pink10, short
$P(\alpha \mid$ **steal**, *null*, `NULL`$)$	ball, the, steals, from, purple8
$P(\alpha \mid$ **turnover**, *null*, `NULL`$)$	to, the, ball, kicks, loses

(a) ROBOCUP

Weight Distribution	Top-5 scoring items
$P(r_i.t \mid$ **temperature**$)$	**windDir**, **sleetChance**, **windSpeed**, **freezingRainChance**, **windChill**
$P(r_i.t \mid$ **windSpeed**$)$	**gust**, **null**, **precipPotential**, **windSpeed**, **snowChance**
$P(r_i.t \mid$ **skyCover**$)$	**temperature**, **skyCover**, **thunderChance**, **null**, **rainChance**
$P(f_i \mid$ temperature.*time*$)$	*min*, *max*, *mean*, *null*, *time*
$P(f_i \mid$ windSpeed.*min*$)$	*max*, *time*, *percent*, *mean*, *null*
$P(f_i \mid$ gust.*max*$)$	*min*, *mean*, *null*, *time*, *max*
$P(\alpha \mid$ skyCover, *percent*, `0-25`$)$	",", clear, mostly, sunny, mid
$P(\alpha \mid$ skyCover, *percent*, `25-50`$)$	",", cloudy, partly, clouds, increasing
$P(\alpha \mid$ rainChance, *mode*, `Definitely`$)$	rain, of, and, the, storms

(b) WEATHERGOV

Weight Distribution	Top-5 scoring items
$P(r_i.t \mid$ **search**$)$	**flight**, **search**, **when**, **day**, **condition**
$P(r_i.t \mid$ **flight**$)$	**search**, **day**, **flight**, **month**, **condition**
$P(r_i.t \mid$ **day**$)$	**when**, **search**, **flight**, **month**, **condition**
$P(\alpha \mid$ **flight**, *to*, `mke`$)$	mitchell, general, international, takeoffs, depart
$P(\alpha \mid$ **search**, *what*, `flight`$)$	I, a, like, to, flight
$P(\alpha \mid$ **search**, *type*, `query`$)$	list, the, me, please, show

(c) ATIS

Table 3: Top-5 scoring items of the multinomial distributions for record rules, field rules and the categorical word rewrite rule of G_{GEN} (see rules (2), (4), and (8) in Table 1, respectively). The first column of each table shows the underlying multinomial distribution for the corresponding rule. For example $P(\alpha \mid$ **pass**, *from*, `purple2`$)$, corresponds to the distribution of emitting word α given the value `purple2` of the field *from* of the record with type **pass**.

from WEATHERGOV, ROBOCUP, and ATIS, respectively. The optimal values for k and β_{LM} for the three domains (when evaluating system performance with BLEU-4) are shown in Table 4.

We conducted two different tuning runs, one for a version of our model that only takes the LM into account (k-BEST-LM; $\beta_{LM} = 1$) and another one where the LM and the DMV are integrated (k-BEST-LM-DMV). As can be seen, optimal values for k are generally larger for k-BEST-LM-DMV. This is probably due to noise introduced by the DMV; as a result, the decoder has to explore the search space more thoroughly. In an effort to investigate the impact of the DMV further, we fixed

k-BEST-LM	k
ROBOCUP	25
WEATHERGOV	15
ATIS	40

(a) Interpolation with LM

k-BEST-LM-DMV	k	β_{LM}
ROBOCUP	85	0.9
WEATHERGOV	65	0.3
ATIS	40	0.6

(b) Interpolation with LM and DMV

Table 4: Optimal values for parameters k and β_{LM} calculated by performing grid search against BLEU-4 on the development set. β_{LM} in Table (a) is set to 1.

$\beta_{LM} = 0$ on the development set and performed a grid search with the DMV on its own. Model performance dropped significantly (by 5–8% BLEU points) which is not entirely surprising given that the DMV alone cannot guarantee fluent output. Its contribution rather rests on capturing more global dependencies outwith the local horizon of the language model.

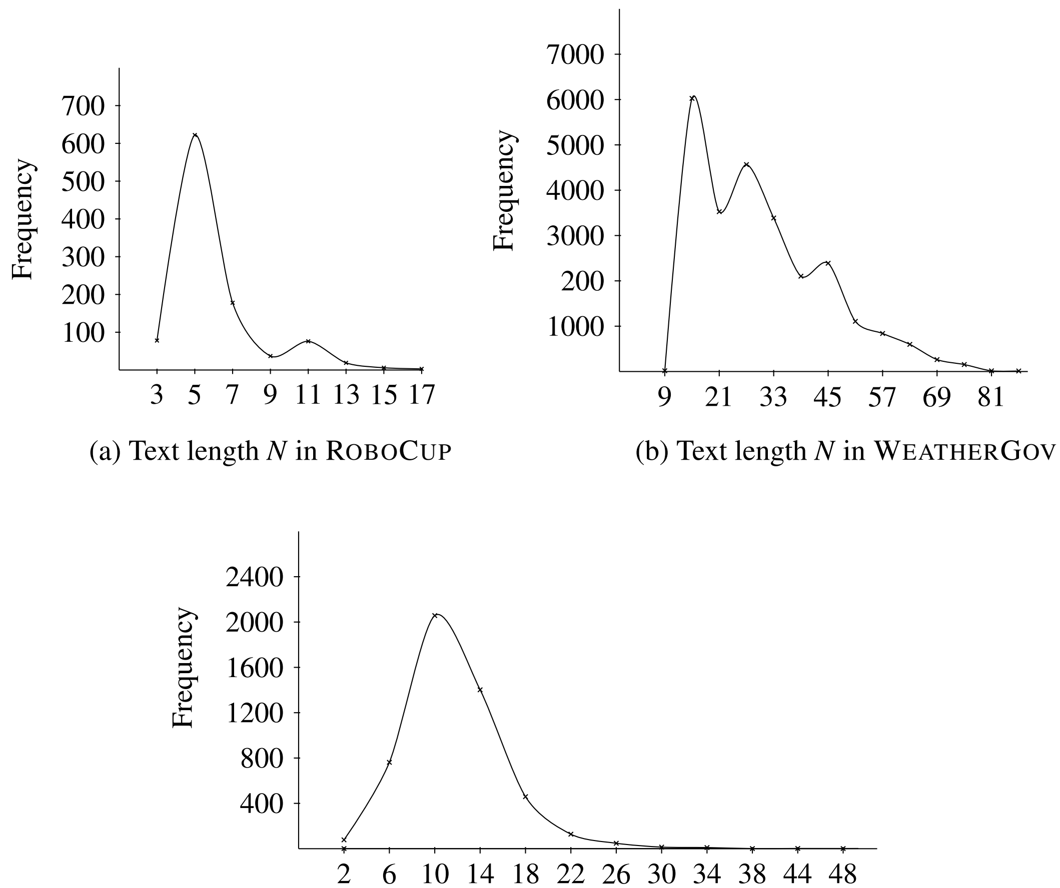
4.3 Determining the Output Length

Unlike other generation systems that operate on the surface realization level with word templates, we emit each word individually in a bottom-up fashion. Therefore, we need to decide on the number of words N we wish to generate before beginning the decoding process. A common approach is to fix N to the average text length of the training set (Banko, Mittal, & Witbrock, 2000). However, this would not be a good choice in our case, since text length does not follow a normal distribution. As shown in Figure 12 the distribution of N across domains is mostly skewed.

To avoid making unwarranted assumptions about our output, we trained a linear regression model that determines the text length individually for each scenario. As input to the model, we used a flattened version of the database, with features being record-field pairs. The underlying idea is that if a scenario contains many records and fields, then we should use more words to express them. In contrast, if the number of records and fields is small, then it is likely that the output is more laconic. In an attempt to capture the number of words needed to communicate specific record-field pairs, we experimented with different types of feature values, e.g., by setting a feature to its actual value (categorical or numerical) or its frequency in the training data. The former scheme worked better in denser datasets, such as WEATHERGOV and ROBOCUP whereas the latter was adopted in ATIS which has a sparser database, as a means to smooth out infrequent values. When trained on the training set and tested on the development set our regression model obtained a correlation coefficient of 0.64 for ROBOCUP, 0.84 for WEATHERGOV, and 0.73 for ATIS (using Pearson's r).

4.4 System Comparison

We evaluated three configurations of our system. A baseline that uses the top scoring derivation in each subgeneration (1-BEST) and two versions of our model that make better use of our decoding algorithm. One version integrates the k-best derivations with a LM (k-BEST-LM), the other version additionally takes the DMV into account (k-BEST-LM-DMV). Preliminary experiments with a model that integrates the k-best derivations with the DMV did not exhibit satisfactory results (see Section 4.2) and we omit them here for the sake of brevity. We compared the output of our models to



(a) Text length N in ROBOCUP

(b) Text length N in WEATHERGOV

(c) Text length N in ATIS

Figure 12: Text length distribution in ROBOCUP, WEATHERGOV, and ATIS (training set).

Angeli et al. (2010) whose approach is closest to ours and state-of-the-art on the WEATHERGOV.[15] For ROBOCUP, we also compared against the best-published results (Kim & Mooney, 2010).

4.5 Evaluation

We evaluated system output automatically, using the BLEU-4 modified precision score (Papineni, Roukos, Ward, & Zhu, 2002) with the human-written text as reference. In addition, we evaluated the generated text via a judgment elicitation study. Participants were presented with a scenario and its corresponding verbalization and were asked to rate the latter along two dimensions: fluency (is the text grammatical and overall understandable?) and semantic correctness (does the meaning conveyed by the text correspond to the database input?). The subjects used a five point rating scale where a high number indicates better performance. We randomly selected 12 documents from the test set (for each domain) and generated output with our models (1-BEST and k-BEST-LM-DMV) and Angeli et al.'s (2010) model. We also included the original text (HUMAN) as gold standard. We thus

15. We are grateful to Gabor Angeli for providing us with the code of his system.

	System	BLEU
JOINT	1-BEST	$8.01^{\triangleright\diamond}$
	k-BEST-LM	24.88^{*}
	k-BEST-LM-DMV	23.14^{*}
FIXED	1-BEST	$10.79^{\triangleright\diamond\circ\dagger}$
	k-BEST-LM	$30.90^{*\dagger}$
	k-BEST-LM-DMV	$29.73^{*\dagger}$
	ANGELI	$28.70^{*\dagger}$
	KIM-MOONEY	$47.27^{*\triangleright\diamond\circ}$

(a) ROBOCUP

System	BLEU
1-BEST	$8.64^{\triangleright\diamond\circ}$
k-BEST-LM	$33.70^{*\diamond\circ}$
k-BEST-LM-DMV	$34.18^{*\triangleright\circ}$
ANGELI	$38.40^{*\triangleright\diamond}$

(b) WEATHERGOV

System	BLEU
1-BEST	$11.85^{\triangleright\diamond\circ}$
k-BEST-LM	$29.30^{*\diamond}$
k-BEST-LM-DMV	$30.37^{*\circ}$
ANGELI	$28.70^{*\diamond}$

(c) ATIS

Table 5: BLEU-4 scores on ROBOCUP, WEATHERGOV, and ATIS (*: significantly different from 1-BEST; $^\circ$: significantly different from ANGELI; $^\triangleright$ significantly different from k-BEST-LM; $^\diamond$: significantly different from k-BEST-LM-DMV; †: significantly different from KIM-MOONEY.

obtained ratings for 48 (12×4) scenario-text pairs for each domain. The study was conducted over the Internet using Amazon Mechanical Turkand involved 305 volunteers (104 for ROBOCUP, 101 for WEATHERGOV, and 100 for ATIS), all self reported native English speakers. Our experimental instructions are given in Appendix A.

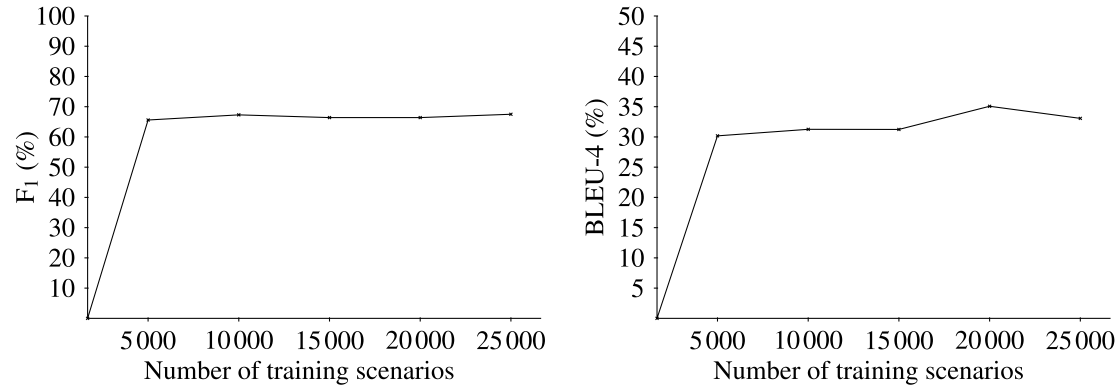
5. Results

We conducted two experiments on the ROBOCUP domain. We first assessed the performance of our generator on joint content selection and surface realization and obtained the results shown in the upper half of Table 5a (see JOINT). In a second experiment we forced the generator to use the gold-standard records from the database. This was necessary in order to compare with previous work (Angeli et al., 2010; Kim & Mooney, 2010).[16] Our results are summarized in lower half of Table 5a (see FIXED).

Overall, our generator performs better than the 1-BEST baseline and comparably to Angeli et al. (2010). k-BEST-LM-DMV is slightly worse than k-BEST-LM. This is due to the fact that sentences in ROBOCUP are very short (their average length is 5.7 words) and as a result our model cannot recover any meaningful dependencies. Using the Wilcoxon signed-rank test we find that differences in BLEU scores among k-BEST-LM-DMV, k-BEST-LM and ANGELI are not statistically significant. Kim and Mooney (2010) significantly outperform these three models and the 1-BEST baseline ($p < 0.01$). This is not entirely surprising, however, as their model requires considerable more supervision (e.g., during parameter initialization) and includes a post-hoc re-ordering component. Finally, we also observe a substantial increase in performance compared to the joint content selection and surface realization setting. This is expected as the generator is faced with an easier task and there is less scope for error.

With regard to WEATHERGOV, our model (k-BEST-LM and k-BEST-LM-DMV) significantly improves over the 1-BEST baseline ($p < 0.01$) but lags behind Angeli et al. (2010) and the difference is

16. Angeli et al. (2010) and Kim and Mooney (2010) fix content selection both at the record and field level. We let our generator select the appropriate fields, since these are at most two per record type and this level of complexity can be easily tackled during decoding.



(a) Alignment

(b) Generation output

Figure 13: Learning curves displaying how the quality of the alignments and generated output vary as a function of the size of the training data.

statistically significant ($p < 0.01$). Since our system emits words based on a language model rather than a template, it displays more freedom in word order and lexical choice, and thus is likelier to produce more *creative* output, sometimes even overly distinct compared to the reference. Dependencies seem to play a more important role here, yielding overall better performance.[17] Interestingly, k-BEST-LM-DMV is significantly better than k-BEST-LM in this domain ($p < 0.01$). Sentences in WEATHERGOV are longer than in ROBOCUP and this allows the k-BEST-LM-DMV to learn dependencies that capture information complementary to the language model.

On ATIS, the k-BEST-LM-DMV model significantly outperforms the 1-BEST ($p < 0.01$) and ANGELI ($p < 0.05$), whereas k-BEST-LM performs comparably. Furthermore, k-BEST-LM-DMV is significantly better than k-BEST-LM ($p < 0.01$). The ATIS domain is the most challenging with respect to surface realization. The vocabulary is larger than ROBOCUP by a factor of 4.3 and WEATHERGOV by a factor of 2.7. Because of the increased vocabulary the model learns richer dependencies which improve its fluency and overall performance.

We also examined the amount of training data required by our model. We performed learning experiments on WEATHERGOV since it contains more training scenarios than ROBOCUP and ATIS and is more challenging with regard to content selection. Figures 13(a) and (b) show how the number of training instances influnces the quality of the alignment and generation output, respectively. We measure alignment F-score following the methodology outlined in the work of Liang et al. (2009) using their gold alignments. The graphs show that 5,000 scenarios are enough for obtaining reasonable alignments and generation output. A very small upward trend can be detected with increasing training instances, however it seems that considerably larger amounts would be required to obtain noticeable improvements.

17. DMV is commonly trained on a sentence-by-sentence basis. In the ROBOCUP and ATIS datasets, each scenario-text pair corresponds to a single sentence. In WEATHERGOV, however, the text may include multiple sentences. In the latter case we trained the DMV on the multi-sentence text without presegmenting it into individual sentences. This non-standard training regime did not seem to pose any difficulty in this domain, as we can safely assume that all examples have the same elided root head, namely "*weather*" (e.g., *The weather* is mostly cloudy, with a low around 30).

	ROBOCUP		WEATHERGOV		ATIS	
System	F	SC	F	SC	F	SC
1-BEST	$2.14^{\diamond\dagger\circ}$	$2.09^{\diamond\dagger\circ}$	$2.25^{\diamond\dagger\circ}$	$2.53^{\diamond\dagger\circ}$	$2.40^{\diamond\dagger\circ}$	$2.49^{\diamond\dagger\circ}$
k-BEST-LM-DMV	4.05^{*}	$3.55^{*\dagger}$	3.89^{*}	3.54^{*}	3.96^{*}	$3.82^{*\circ}$
ANGELI	4.01^{*}	$3.47^{*\dagger}$	3.82^{*}	3.72^{*}	3.86^{*}	$3.31^{*\dagger\diamond}$
HUMAN	4.17^{*}	$3.97^{*\diamond\circ}$	4.01^{*}	3.58^{*}	4.16^{*}	$3.96^{*\circ}$

Table 6: Mean ratings for fluency (F) and semantic correctness (SC) on system output elicited by humans on ROBOCUP, WEATHERGOV, and ATIS (*: significantly different from 1-BEST; $^\circ$: significantly different from ANGELI; $^\diamond$: significantly different from *k*-BEST-LM-DMV; †: significantly different from HUMAN).

The results of our human evaluation study are shown in Table 6. We report mean ratings for each system and the gold-standard human authored text. Our experimental participants rated the output on two dimensions, namely fluency (F) and semantic correctness (SC). We elicited judgments only for *k*-BEST-LM-DMV as it generally performed better than *k*-BEST-LM in our automatic evaluation (see Table 5). We carried out an Analysis of Variance (ANOVA) to examine the effect of system type (1-BEST, *k*-BEST-LM-DMV, ANGELI, and HUMAN) on the fluency and semantic correctness ratings. We used Tukey's Honestly Significant differences (HSD) test, as explained by Yandell (1997) to assess whether means differences are statistically significant.

On all three domains our system (*k*-BEST-LM-DMV) is significantly better than the 1-BEST baseline ($a < 0.01$) in terms of fluency. Our output is indistinguishable from the gold-standard (HUMAN) and ANGELI (pair-wise differences among *k*-BEST-LM-DMV, ANGELI and HUMAN are not statistically significant). With respect to semantic correctness, on ROBOCUP, *k*-BEST-LM-DMV is significantly better than 1-BEST ($a < 0.01$) but significantly worse than HUMAN ($a < 0.01$). Although the ratings for *k*-BEST-LM-DMV are numerically higher than ANGELI, the difference is not statistically significant. ANGELI is also significantly worse than HUMAN ($a < 0.01$). On WEATHERGOV, the semantic correctness of *k*-BEST-LM-DMV and ANGELI is not significantly different. These two systems are also indistinguishable from HUMAN. On ATIS, *k*-BEST-LM-DMV is the best performing model with respect to semantic correctness. It is significantly better than 1-BEST and ANGELI ($a < 0.01$) but not significantly different from HUMAN.

In sum, we observe that performance improves when *k*-best derivations are taken into account (the 1-BEST system is consistently worse). Our results also show that taking dependency-based information into account boosts model performance over and above what can be achieved with a language model. Our model is on par with ANGELI on ROBOCUP and WEATHERGOV but performs better on ATIS when evaluated both automatically and by humans. Error analysis suggests that a reason for ANGELI's poorer performance on ATIS might be its inability to create good quality surface templates. This is due to the lack of sufficient data and the fact that templates cannot fully express the same database configurations in many different ways. This is especially true for ATIS which consists of transcriptions of spontaneous spoken utterances and the same meaning can be rendered in many different ways. For example, the phrases "*show me the flights*", "*what are the flights*", "*which flights*", and "*please can you give me the flights*", all convey the exact same meaning stemming from a **Search** record.

Our model learns domain specific conventions about "how to say" and "what to say" from data, without any hand-engineering or manual annotation. Porting the system to a different domain is

Input:

Bad Pass	
from	*to*
pink11	purple5

1-BEST: pink11 pass purple5 purple5 pink11 pass purple5 purple5 purple5

k-BEST-LM-DMV: pink11 made a pass that was intercepted by purple5

ANGELI: pink11 made a bad pass that missed its target and was picked up by purple5

HUMAN: pink11 tries to pass but was intercepted by purple5

(a) ROBOCUP

Input:

Temperature			
time	*min*	*mean*	*max*
06:00-21:00	32	39	46

Cloud Sky Cover	
time	*percent (%)*
06:00-21:00	75-100

Wind Speed			
time	*min*	*mean*	*max*
06:00-21:00	6	7	10

Wind Direction	
time	*mode*
06:00-21:00	SE

1-BEST: Near 46. Near 46. Near 46. Near 46. Near 46. With near 46. Southeast wind.

k-BEST-LM-DMV: Mostly cloudy, with a high near 46. South southeast wind between 6 and 10 mph.

ANGELI : A chance of rain or drizzle, with a high near 46. Southeast wind between 6 and 10 mph. mph. Chance of precipitation is 60%.

HUMAN: Mostly cloudy, with a high near 46. South southeast wind between 6 and 10 mph.

(b) WEATHERGOV

Input:

Flight	
from	*to*
milwaukee	phoenix

Day	
day	*dep/ar/ret*
saturday	departure

Search	
type	*what*
query	flight

1-BEST: Milwaukee Phoenix on Saturday on Saturday on Saturday on Saturday

k-BEST-LM-DMV: Show me the flights from Milwuakee to Phoenix on Saturday

ANGELI : Show me the flights between Milwuakee and Phoenix on Saturday

HUMAN: Milwuakee to Phoenix on Saturday

(c) ATIS

Figure 14: Example output on (a) sportscasting, (b) weather forecasting, and (c) air travel domains with correct content selection.

straightforward, assuming a database and corresponding (unaligned) text. As long as the database obeys the structure of the grammar G_{GEN}, we need only retrain the model to obtain the weights of the grammar rules; in addition, the system requires a domain specific language model and optionally

Gold:

Temperature

time	*min*	*mean*	*max*
06:00-21:00	30	38	44

Cloud Sky Cover

time	*percent (%)*
06:00-21:00	75-100

Chance of Rain

Time	**Mode**
06:00-21:00	Slight Chance

Wind Speed

time	*min*	*mean*	*max*
06:00-21:00	6	6	7

Wind Direction

time	*mode*
06:00-21:00	ENE

Precipitation Potential (%)

time	*min*	*mean*	*max*
06:00-21:00	9	20	35

Output: A 40 percent chance of showers before 10am. Mostly cloudy, with a high near 44. East northeast wind around 7 mph.

(a) Gold standard content selection and its verbalization

Content Selection:

Temperature

time	*min*	*mean*	*max*
06:00-21:00	30	38	44

Cloud Sky Cover

time	*percent (%)*
06:00-21:00	75-100

Chance of Rain

time	*mode*
06:00-09:00	Chance

Wind Speed

time	*min*	*mean*	*max*
06:00-21:00	6	6	7

Wind Direction

time	*mode*
06:00-21:00	ENE

Chance of Thunderstorm

time	*mode*
06:00-13:00	--
13:00-21:00	--

Output: A chance of showers. Patchy fog before noon. Mostly cloudy, with a high near 44. East wind between 6 and 7 mph.

(b) *k*-BEST-LM-DMV content selection

Content Selection:

Temperature

time	*min*	*mean*	*max*
06:00-21:00	30	38	44

Precipitation Potential (%)

time	*min*	*mean*	*max*
06:00-21:00	9	20	35

Chance of Rain

time	*mode*
06:00-09:00	Chance

Wind Speed

time	*min*	*mean*	*max*
06:00-21:00	6	6	7

Wind Direction

time	*mode*
06:00-21:00	ENE

Chance of Thunderstorm

time	*mode*
06:00-21:00	--

Output: A chance of showers. Patchy fog before noon. Mostly cloudy, with a high near 44. East wind between 6 and 7 mph. Chance of precipitation is 35%

(c) ANGELI content selection

Figure 15: Example output on WEATHERGOV domain with incorrect content selection (in gray).

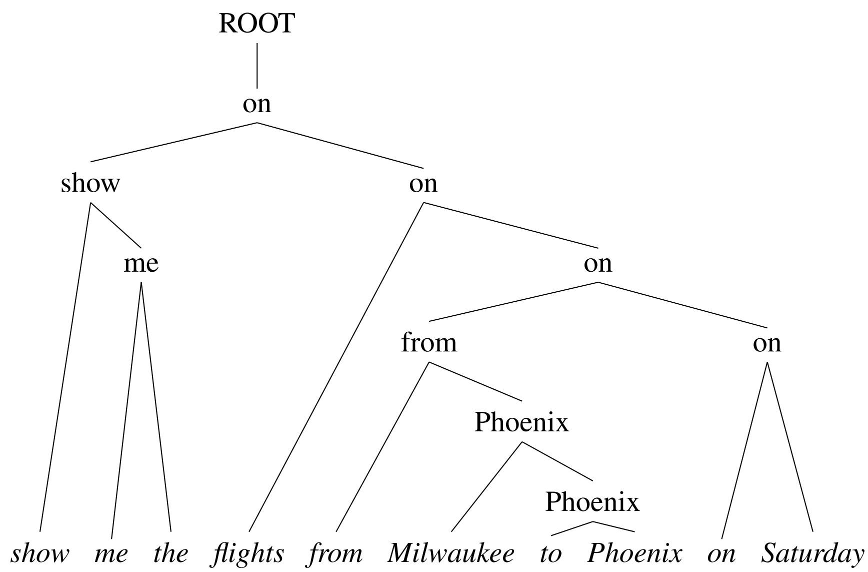


Figure 16: Dependency structure for the sentence *Show me the flights from Milwaukee to Phoenix on Sunday* as generated by k-BEST-LM-DMV (see Figure 14c). Intermediate nodes in the tree denote the head words of each subtree.

information about heads and their dependents which the DMV learns in an unsupervised fashion. In the latter case, we also need to tune the hyperparameter β_{LM}, and in both cases k. Note, that fine-tuning k becomes less important when integrating with a language model only. As we explain in Section 4.2, the DMV possibly introduces noise, therefore we have to modulate k more carefully so as to allow the decoder to search in a bigger space.

Examples of system output with correct content selection at the record level are given in Figure 14. Note that in the case of ROBOCUP, content selection is fixed to the gold standard. As can be seen, the generated text is close to the human authored text. Also note that the output of our system improves considerably when taking k-best derivations into account (compare 1-BEST and k-BEST-LM-DMV in the figure). Figure 15a shows examples with incorrect content selection at the record level for the WEATHERGOV domain. Figure 15a shows the gold standard content selection and its corresponding verbalization. Figures 15b and 15c show the output of the k-BEST-LM-DMV system and ANGELI. Tables in black denote record selection identical to the gold standard, whereas tables in grey denote false positive recall. k-BEST-LM-DMV identifies an incorrect value for the *Mode* field in the **Chance of Rain** record; in addition, it fails to select the **Precipitation Potential (%)** record altogether. The former mistake does not affect the correctness of the generator's output, whereas the latter does (i.e., it fails to mention the exact likelihood of rain, 40% in the gold standard and 35% in ANGELI's output). Finally, Figure 16 shows the dependency structure our model produced for the sentence *Show me the flights from Milwaukee to Phoenix on Saturday* from Figure 14c; notice the long range dependency between *flights* and *on*, which would otherwise be inaccessible to a language model.

6. Conclusions

We have presented an end-to-end generation system that performs content selection and surface realization simultaneously. Central to our approach is the encoding of generation as a parsing problem. We reformulate the input (a set of database records and text describing some of them) as a PCFG and show how to approximately find the best generated string licensed by the grammar. We evaluated our model on three domains (ROBOCUP, WEATHERGOV, ATIS) and showed that it is able to obtain performance comparable or superior to the state-of-the-art. Our experiments were also designed to assess several aspects of the proposed framework such as the use of k-best decoding and the intersection of our grammar with multiple information sources. We observed that k-best decoding is essential to producing good quality output. Across domains, performance increases by a factor of at least two when multiple derivations are taken into account. In addition, intersecting the grammar with dependency-based information seems to capture syntactic information complementary to the language model. We argue that our approach is computationally efficient and viable in practical applications. Outwith generation, we hope that some of the work described here might be of relevance to other fields such as summarization or machine translation.

Future extensions are many and varied. An obvious extension concerns porting the framework to more challenging domains with richer vocabulary and longer texts (e.g., product descriptions, user manuals, sports summaries). A related question is how to extend the PCFG-based approach advocated here so as to capture discourse-level document structure. Other future directions involve exploiting the information available in the database more directly. Our model takes into account the k-best derivations at decoding time, however inspection of these indicates that it often fails to select the best one. Initial work (Konstas & Lapata, 2012) shows that the model presented here can be adapted to use forest reranking, a technique that approximately reranks a packed forest of exponentially many derivations (Huang, 2008). The reranker is essentially a structured perceptron (Collins, 2002) enriched with local and non-local features. It therefore allows to explicitly model dependencies across fields, records, and their interactions.

Finally, although not the focus of this paper, it is worth pointing out that the model described here can also perform semantic parsing, i.e., convert text into a formal meaning representation. This can be done trivially by modifying the grammar in Table 1. Instead of observing words as terminals (rules (8) and (9)), we observe values of fields, given a particular word w, field, and record:

$$\mathrm{W}(r, r.f) \rightarrow f.v \qquad [P(f.v \,|\, r, r.f, f.t, w)]$$

$$\mathrm{W}(r, r.f) \rightarrow \mathrm{gen}(w) \qquad [P(\mathrm{gen}(w).mode \,|\, r, r.f, f.t{=}int)]$$

During decoding, the prior $p(g)$ in equation (4) becomes $p(f.v)$ and can be naively obtained by creating an n-gram language model over the alignments between the meaning representations and the text. Such alignments are in principle hidden but could be estimated using the model of Liang et al. (2009).

Acknowledgments

We are grateful to the anonymous referees whose feedback helped to substantially improve the present paper. Thanks to Luke Zettlemoyer and Tom Kwiatkowski for their help with the ATIS dataset as well as Giorgio Satta and Frank Keller for helpful comments and suggestions. We also thank the members of the Probabilistic Models reading group at the University of Edinburgh for their feedback. A preliminary version of this work was published in the proceedings of NAACL 2012.

Appendix A. Experimental Instructions

A.1 Instructions

In this experiment you will be given tables that contain some facts about the weather (e.g., **Temperature**, **Chance of Rain**, **Wind Direction**, **Cloud Coverage** and so on) and their translation in natural language. Example 1 below tabulates such weather related information and its translation as *Rainy with a high near 47. Windy, with an east wind between 5 and 15 mph.*

Example 1

Category	Fields		
Temperature	time: 17.00–06.00(+1 day)	min: 30	mean: 40 max: 47
Wind Direction	time: 17.00–06.00(+1 day)	mode: SE	
Cloud Sky Cover	time: 17.00–06.00(+1 day)	percent: 25–50	
Chance of Rain	time: 17.00–21.00	mode: Likely	
Rainy with a high near 47. Windy , with an east wind between 5 and 15 mph.			

Each row in the table contains a different weather-related event. The first row talks about temperature, the second one about wind direction, etc. Different event types instantiate different fields. For example, **Temperature** has four fields, *time*, *min*, *mean*, and *max*. Fields in turn have values, which can be either numbers (e.g., `47` degrees Fahrenheit for the event Temperature), or words (e.g., `Likely` or `Slight Chance` for the event Chance of Rain).

More specifically, you should read the above table as follows. For **Temperature**, the field *time* and its value `17.00-06.00(+1 day)` refers to temperatures measured between 5pm and 6am of the following day. The minimum temperature recorded for that time period is `30` degrees Fahrenheit (field *min*), the maximum is `47` degrees (field *max*) and on average the temperature is 40 degrees (field *mean*). For the same time period, the wind will blow from a south east direction (the *mode* of **Wind Direction** is `SE`). 25–50% of the sky will be covered with clouds (see field *percent* with value `25-50` in **Cloud Sky Cover**), which may be interpreted as a slightly cloudy outlook. Finally, from 5pm to 9pm it is likely to rain, as indicated by the `mode` field and its value `Likely` for the **Chance of Rain** event.

Note that all temperature values are in the Fahrenheit scale. The Fahrenheit scale is an alternative temperature scale to Celsius, proposed in 1724 by the physicist Daniel Gabriel Fahrenheit. The formula that converts Fahrenheit degrees to Celsius is [F] = [C] $\times \frac{9}{5} + 32$. So, for instance, -1 C = 30 F. Also note, the measure of speed used throughout the experiment is miles per hour, mph for short.

All natural language translations have been generated by a computer program. Your task is to rate the translations on two dimensions, namely Fluency and Semantic Correctness on a scale from

1 to 5. As far as Fluency is concerned, you should judge whether the translation is grammatical and in well-formed English or just gibberish. If the translation is grammatical, then you should rate it high in terms of fluency. If there is a lot of repetition in the translation or if it seems like word salad, then you should give it a low number.

Semantic Correctness refers to the meaning conveyed by the translation and whether it corresponds to what is reported in the tabular data. In other words, does the translation convey the same content as the table or not? If the translation has nothing to do with the categories, fields or values described in the table, you should probably give it a low number for Semantic Correctness. If the translation captures most of the information listed in the table, then you should give it a high number. Bear in mind that slight numerical deviations are normal and should not be penalized (e.g., it is common for weather forecasters to round wind speed values to the closest 5, i.e., '50 mph' instead of '47 mph').

A.2 Rating Examples

In Example 1, you would probably give the translation a high score for Fluency (e.g., 4 or 5), since it is coherent and does not contain any grammatical errors. However, you should give it a low score for Semantic Correctness (e.g., 1–3), because it conveys information that is not in the table. For example, '*windy*' and '*wind between 5 and 15 mph*' both relate to wind speed but are not mentioned in the table. Let us now consider the following example:

Example 2

Category	Fields		
Temperature	time: 17.00–06.00(+1 day)	min: 40	mean: 45 max: 50
Wind Direction	time: 17.00–06.00(+1 day)	mode: S	
Wind Speed	time: 17.00– 06.00(+1 day)	min: 5	mean: 7 max: 15
Cloud Sky Cover	time: 17.00–06.00(+1 day)	percent: 0–25	
Sunny, with a low around 40. South wind between 5 and 15 mph.			

Here, you should give the translation high scores on both dimensions, namely Fluency and Semantic Correctness. The text is grammatical and succinctly describes the content of the table. For example, 4 or 5 would be appropriate numbers.

Example 3

Category	Fields		
Temperature	time: 17.00–06.00(+1 day)	min: 30	mean: 40 max:47
Wind Direction	time: 17.00–06.00(+1 day)	mode: ESE	
Around 40. Around 40. Around 40. East wind.			

In example 3, the translation scores poorly on Fluency and Semantic Correctness. The text has many repetitions and there is no clear correspondence between the translation and the table. '*around 40*' probably refers to the temperature, but it is not at all clear from the context of the text. '*east wind*' again refers to wind direction, but it is missing a verb or a preposition that would relate it to the weather outlook. Appropriate scores for both dimensions would be 1 or 2.

Finally, while judging the translation pay attention to the values of the fields in the table in addition to the event categories. For example, you may have an event Chance of Rain with a value `None` in the *mode* field. This means that it is not likely to rain, and you should penalize any mention of rain in the text, unless there is another event Chance of Rain for a different time period with a different value in the mode field.

A.3 Rating Procedure

Before you start the experiment below you will be asked to enter your personal details. Next, you will be presented with 15 table-translation pairs to evaluate in the manner described above. You will be shown one pair at a time. Once you finish with your rating, click the button at the bottom right to advance to the next response.

Things to remember:

- If you are unsure how to rate a translation, click on the top right of your window the Help link. You may also leave it open during the course of the experiment as a reference.
- Higher numbers represent a positive opinion of the translation and lower numbers a negative one.
- Do not spend too long analyzing the translations; you should be able to rate them once you have read them for the first time.
- There is no right or wrong answer, so use your own judgment when rating each translation.

A.4 Personal Details

As part of the experiment we will ask you for a couple of personal details. This information will be treated confidentially and will not be made available to a third party. In addition, none of your responses will be associated with your name in any way. We will ask you to supply the following information.

- Your name and email address.
- Your age and sex.
- To specify, under 'Language Region', the place (city, region/state/province, country) where you have learnt your first language.
- To enter the code provided at the end of the experiment into the Mechanical Turk HIT.

References

Amazon Mechanical Turk (2012). Retrieved from `https://www.mturk.com`..

Angeli, G., Liang, P., & Klein, D. (2010). A simple domain-independent probabilistic approach to generation. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pp. 502–512, Cambridge, MA.

Banko, M., Mittal, V. O., & Witbrock, M. J. (2000). Headline generation based on statistical translation. In *Proceedings of Association for Computational Linguistics*, pp. 318–325, Hong Kong.

Barzilay, R., & Lapata, M. (2005). Collective content selection for concept-to-text generation. In *Proceedings of Human Language Technology and Empirical Methods in Natural Language Processing*, pp. 331–338, Vancouver, British Columbia.

Belz, A. (2008). Automatic generation of weather forecast texts using comprehensive probabilistic generation-space models. *Natural Language Engineering*, *14*(4), 431–455.

Charniak, E., & Johnson, M. (2005). Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*, pp. 173–180, Ann Arbor, Michigan.

Chen, D. L., & Mooney, R. J. (2008). Learning to sportscast: A test of grounded language acquisition. In *Proceedings of International Conference on Machine Learning*, pp. 128–135, Helsinki, Finland.

Chiang, D. (2007). Hierarchical phrase-based translation. *Computational Linguistics*, *33*(2), 201–228.

Collins, M. (2002). Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*, pp. 1–8, Philadelphia, Pennsylvania.

Dahl, D. A., Bates, M., Brown, M., Fisher, W., Hunicke-Smith, K., Pallett, D., Pao, C., Rudnicky, A., & Shriberg, E. (1994). Expanding the scope of the ATIS task: the ATIS-3 corpus. In *Proceedings of the Workshop on Human Language Technology*, pp. 43–48, Plainsboro, New Jersey.

Dale, R., Geldof, S., & Prost, J.-P. (2003). Coral: Using natural language generation for navigational assistance. In *Proceedings of the 26th Australasian Computer Science Conference*, pp. 35–44, Adelaide, Australia.

de Gispert, A., Iglesias, G., Blackwood, G., Banga, E. R., & Byrne, W. (2010). Hierarchical phrase-based translation with weighted finite-state transducers and shallow-n grammars. *Computational Linguistics*, *36*(3), 505–533.

Duboue, P. A., & McKeown, K. R. (2002). Content planner construction via evolutionary algorithms and a corpus-based fitness function. In *Proceedings of International Natural Language Generation*, pp. 89–96, Ramapo Mountains, NY.

Gallo, G., Longo, G., Pallottino, S., & Nguyen, S. (1993). Directed hypergraphs and applications. *Discrete Applied Mathematics*, *42*, 177–201.

Goldberg, E., Driedger, N., & Kittredge, R. (1994). Using natural-language processing to produce weather forecasts. *IEEE Expert*, *9*(2), 45–53.

Good, I. J. (1953). The population frequencies of species and the estimation of population parameters. *Biometrika*, *40*(3/4), pp. 237–264.

Goodman, J. (1999). Semiring parsing. *Computational Linguistics*, *25*(4), 573–605.

Green, N. (2006). Generation of biomedical arguments for lay readers. In *Proceedings of the 5th International Natural Language Generation Conference*, pp. 114–121, Sydney, Australia.

Huang, L. (2008). Forest reranking: Discriminative parsing with non-local features. In *Proceedings of ACL-08: HLT*, pp. 586–594, Columbus, Ohio.

Huang, L., & Chiang, D. (2005). Better k-best parsing. In *Proceedings of the 9th International Workshop on Parsing Technology*, pp. 53–64, Vancouver, British Columbia.

Huang, L., & Chiang, D. (2007). Forest rescoring: Faster decoding with integrated language models. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pp. 144–151, Prague, Czech Republic.

Iglesias, G., Allauzen, C., Byrne, W., de Gispert, A., & Riley, M. (2011). Hierarchical phrase-based translation representations. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pp. 1373–1383, Edinburgh, Scotland, UK. Association for Computational Linguistics.

Kasami, T. (1965). An efficient recognition and syntax analysis algorithm for context-free languages. Tech. rep. AFCRL-65-758, Air Force Cambridge Research Lab, Bedford, Massachusetts.

Kim, J., & Mooney, R. (2010). Generative alignment and semantic parsing for learning from ambiguous supervision. In *Proceedings of the 23rd Conference on Computational Linguistics*, pp. 543–551, Beijing, China.

Klein, D., & Manning, C. (2004). Corpus-based induction of syntactic structure: Models of dependency and constituency. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics*, pp. 478–485, Barcelona, Spain.

Klein, D., & Manning, C. D. (2001). Parsing and hypergraphs. In *Proceedings of the 7th International Workshop on Parsing Technologies*, pp. 123–134, Beijing, China.

Konstas, I. (2013). ATIS dataset retrieved from `http://homepages.inf.ed.ac.uk/ikonstas/index.php?page=resources`..

Konstas, I., & Lapata, M. (2012). Concept-to-text generation via discriminative reranking. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pp. 369–378, Jeju, South Korea.

Li, Z., & Eisner, J. (2009). First- and second-order expectation semirings with applications to minimum-risk training on translation forests. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pp. 40–51, Suntec, Singapore.

Liang, P., Bouchard-Côté, A., Klein, D., & Taskar, B. (2006). An end-to-end discriminative approach to machine translation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics*, pp. 761–768, Sydney, Australia.

Liang, P., Jordan, M., & Klein, D. (2009). Learning semantic correspondences with less supervision. In *roceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pp. 91–99, Suntec, Singapore.

Lu, W., & Ng, H. T. (2011). A probabilistic forest-to-string model for language generation from typed lambda calculus expressions. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pp. 1611–1622, Edinburgh, Scotland, UK.

Marcus, M. P., Marcinkiewicz, M. A., & Santorini, B. (1993). Building a large annotated corpus of English: the Penn treebank. *Comput. Linguist.*, *19*(2), 313–330.

Nederhof, M.-J., & Satta, G. (2004). The language intersection problem for non-recursive context-free grammars. *Information and Computation*, *192*(2), 172 – 184.

Och, F. J. (2003). Minimum error rate training in statistical machine translation. In *Proceedings of the Annual Meeting on Association for Computational Linguistics*, pp. 160–167, Sapporo, Japan.

Papineni, K., Roukos, S., Ward, T., & Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, pp. 311–318, Philadelphia, Pennsylvania.

Ratnaparkhi, A. (2002). Trainable approaches to surface natural language generation and their application to conversational dialog systems. *Computer Speech & Language*, *16*(3-4), 435–455.

Reiter, E., & Dale, R. (2000). *Building natural language generation systems*. Cambridge University Press, New York, NY.

Reiter, E., Sripada, S., Hunter, J., & Davy, I. (2005a). Choosing words in computer-generated weather forecasts. *Artificial Intelligence*, *167*, 137–169.

Reiter, E., Sripada, S., Hunter, J., Yu, J., & Davy, I. (2005b). Choosing words in computer-generated weather forecasts. *Artificial Intelligence*, *167*, 137–169.

Shieber, S. M., Schabes, Y., & Pereira, F. C. N. (1995). Principles and implementation of deductive parsing. *Logic Programming*, *24*, 3–36.

Sripada, S. G., Reiter, E., Hunter, J., & Yu, J. (2003). Generating English summaries of time series data using the gricean maxims. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 187–196. ACM Press.

Stolcke, A. (2002). SRILM – an extensible language modeling toolkit. In Hansen, J. H. L., & Pellom, B. L. (Eds.), *Proceedings of the 7th International Conference on Spoken Language Processing*, pp. 901–904, Denver, Colorado. ISCA.

Toutanova, K., Klein, D., Manning, C. D., & Singer, Y. (2003). Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, pp. 173–180, Edmonton, Canada.

Turner, R., Sripada, Y., & Reiter, E. (2009). Generating approximate geographic descriptions. In *Proceedings of the 12th European Workshop on Natural Language Generation*, pp. 42–49, Athens, Greece.

Wong, Y. W., & Mooney, R. (2007). Generation by inverting a semantic parser that uses statistical machine translation. In *Proceedings of the Human Language Technology and the Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 172–179, Rochester, NY.

Yandell, B. S. (1997). *Practical Data Analysis for Designed Experiments*. Chapman & Hall/CRC.

Younger, D. H. (1967). Recognition and parsing for context-free languages in time n^3. *Information and Control*, *10*(2), 189–208.

Zettlemoyer, L., & Collins, M. (2007). Online learning of relaxed CCG grammars for parsing to logical form. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pp. 678–687, Prague, Czech Republic.