Joint Models for Concept-to-text Generation

Ioannis Konstas

Doctor of Philosophy
Institute for Language, Cognition and Computation
School of Informatics
University of Edinburgh
2014

Abstract

Much of the data found on the world wide web is in numeric, tabular, or other non-textual format (e.g., weather forecast tables, stock market charts, live sensor feeds), and thus inaccessible to non-experts or laypersons. However, most conventional search engines and natural language processing tools (e.g., summarisers) can only handle textual input. As a result, data in non-textual form remains largely inaccessible. Concept-to-text generation refers to the task of automatically producing textual output from non-linguistic input, and holds promise for rendering non-linguistic data widely accessible. Several successful generation systems have been produced in the past twenty years. They mostly rely on human-crafted rules or expert-driven grammars, implement a pipeline architecture, and usually operate in a single domain.

In this thesis, we present several novel statistical models that take as input a set of database records and generate a description of them in natural language text. Our unique idea is to combine the processes of structuring a document (document planning), deciding what to say (content selection) and choosing the specific words and syntactic constructs specifying how to say it (lexicalisation and surface realisation), in a uniform joint manner. 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). This joint representation allows individual processes (i.e., document planning, content selection, and surface realisation) to communicate and influence each other naturally.

We recast generation as the task of finding the best derivation tree for a set of input database records and our grammar, and describe several algorithms for decoding in this framework that allows to intersect the grammar with additional information capturing fluency and syntactic well-formedness constraints. We implement our generators using the hypergraph framework. Contrary to traditional systems, we learn all the necessary document, structural and linguistic knowledge from unannotated data. Additionally, we explore a discriminative reranking approach on the hypergraph representation of our model, by including more refined content selection features. Central to our approach is the idea of porting our models to various domains; we experimented on four widely different domains, namely sportscasting, weather forecast generation, booking flights, and troubleshooting guides. The performance of our systems is competitive and often superior compared to state-of-the-art systems that use domain specific constraints, explicit feature engineering or labelled data.

Lay Summary

Much of the data found on the world wide web is in numeric, tabular, or other non-textual format (e.g., weather forecast tables, stock market charts, live sensor feeds), and thus inaccessible to non-experts or laypersons. However, most conventional search engines and natural language processing tools (e.g., summarisers) can only handle textual input. As a result, data in non-textual form remains largely inaccessible. Concept-to-text generation refers to the task of automatically producing textual output from non-linguistic input, and holds promise for rendering non-linguistic data widely accessible. Several successful generation systems have been produced in the past twenty years. They mostly rely on human-crafted rules, implement a pipeline architecture, and usually operate in a single domain, such as weather forecasts, virtual museums, etc.

In this thesis, we present several novel systems that take as input non-textual data, (e.g., numeric tabular data in a weather forecast) and generate a description of it in natural language text. Contrary to older systems that relied on human-crafted rules, we acquire all the necessary information we need from the input data. We thus learn which parts of the input to include, in what specific order, and which exact words to choose in order to describe them, from frequently occurring patterns in the input. Our unique idea is to combine the processes of creating a layout of the document, deciding what to say and choosing the specific words and syntactic constructs specifying how to say it, in a uniform joint manner. Traditionally, generation systems break down these processes into separate computer programs, and tackle each of them individually and independently. We chose instead to represent all of them globally in a single framework, allowing each process to communicate and influence each other naturally. We used our systems in several different domains, namely sportscasting, weather forecast generation, booking flights, and troubleshooting guides. The output of our systems is competitive and often superior to the output of other similar systems that also acquire their knowledge from data, but include some form of human intervention (e.g., in the weather forecast domain, they included rules that specifically instruct to mention rain, if the corresponding percentage of rain is particularly high in the input data).

Acknowledgements

First of all I would like to thank my supervisor, Mirella Lapata. She literally shaped my way of thinking, observing, and writing. Her advice to always keep a critical mind and question everything has strengthened my research capabilities, and certainly influenced my approach to computational linguistics. Her meticulous, thorough and prompt feedback kept me on the right track from the beginning till the end, leaving little space for doubt or worry. But most importantly, she gave me the freedom to work on things that fascinated me most, while at the same time making sure they are manageable and useful. Needless to say that her vibrant personality and enthusiastic character has had the most lasting impression on me; our frequent meetings are something that I will definitely miss. I would also like to thank Charles Sutton, my second supervisor, for his very helpful discussions and brilliant suggestions, especially on technical issues. I also wish to thank my two examiners, Stephen Clark and Jon Oberlander for their insightful comments and interesting discussion we had during my viva examination.

Many thanks go to my colleagues in the Probabilistic Models of Language (Prob-Models) reading group; their input and feedback has proved valuable for shaping many ideas included in this thesis. More specifically, I would like to personally thank Christos Christodoulopoulos, Tom Kwiatkowski, Yansong Feng, Greg Coppola, Kristian Woodsend, Carina Silberer, William Blacoe, Siva Reddy, and last but not least, Dimitrios Milios. I would also like to thank Giorgio Satta, Luke Zettlemoyer, Michael Collins, Frank Keller, and Oliver Lemon for their helpful advice and feedback.

My deepest thanks also go to my brother and parents for their love and support all these years. Finally, my warmest gratitude goes to Maria Eirini Politou for being loving, patient and kind enough to support me, both during the good and stressful times, throughout my PhD journey.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Ioannis Konstas*)

Table of Contents

Chapter 1

Introduction

1.1 Motivation

The explosion of the world wide web (WWW) has generated unprecedented amounts of data and made them available to countless end-users. The way we access and process information has been revolutionised multiple times: in the first years of its use the WWW worked more as a replacement to libraries and galleries, hosting mostly static content. In the last ten years, the advent of Web 2.0 and social media changed the WWW to a more interactive, and user-centric environment. Consequently, this development increased the amount of online user-generated data available online. Undeniably, a great part of the success of the WWW is due to information retrieval and natural language processing (NLP) applications, which allow us to find and structure the content available. Examples of such applications include various search engines, automatic query suggestion, summarisation, and so on.

However, much of this content is in numeric, tabular form, or in a format otherwise inaccessible to non-expert or casual users (see for example the density of the tabular format of a weather forecast in Figure 1.1, and the clutter of are the aggregated graphs in a stock market trend chart in Figure 1.2). This content is represented in many different forms such as ontologies, databases, and formal logic, stored in various formats and languages (Web Ontology Language - OWL, some sort of Structured Query Language - SQL, lambda calculus) and is presented to the user in equally many ways (semantic graphs, database tables, spreadsheet files, charts). An immediate consequence is that search engines and existing NLP tools cannot directly handle this type of non-linguistic information. Acquiring the appropriate expertise for reading and understanding such data is costly and time-consuming. From the expert user's perspective, processing large amounts of similar-looking numerical data in a mechanical way quickly becomes a mundane, laborious task and can occasionally lead to mistakes and errors. These are some reasons for the growing interest in the NLP community in the past few decades, to develop concept-to-text generation[1] systems that automatically produce textual output in natural language from non-linguistic input (Reiter and Dale, 2000).

A typical example application of a concept-to-generation system is to automatically generate a weather forecast, from time series data representing various meteorological measurements such as temperature and wind speed, as shown in Figure 1.1. Several research studies, even commercial systems, have been successfully developed since the early 1990s for this particular domain (e.g., FOG; Goldberg et al. 1994, SUMTIME-MOUSAM; Reiter et al. 2005, inter alia). Another example application is to generate user-specific descriptions of artefacts in the context of a digital museum, from information stored in a database. M-PIRO (Isard et al., 2003) is such a system that produces text of different detail given the age and background of the audience (children, adults, or experts). Figure 1.3 shows an example of a typical input and output text catered for an adult visitor. Similar systems in less common domains also exist such as, e.g., the generation of letters to help people stop smoking (STOP; Reiter et al. 2003); Figure 1.4 illustrates an excerpt from a personalised document, generated automatically given the filled in questionnaire from its recipient. Finally, BabyTalk (Portet et al., 2009) (see Figure 1.5) is a generation system that summarises clinical data about premature babies in neonatal intensive care units. Given raw sensor data and a set of actions performed by medical staff, it outputs a summary of the health of the baby personalised individually for doctors, nurses and family.

Despite the prominent success of concept-to-text generation systems, there are a few considerable downsides with the techniques they adopt. First of all, adapting individual systems to an entirely different domain or porting them to another language, practically entails re-engineering and testing a good part of them again. The reason is simply because most of the current generation systems rely on rules and hand-crafted specifications (such as in-domain grammars) that collectively define the following modular decisions: which parts of the input should be selected (*content planning*), how they should be combined into textual form (*sentence planning*), and how

[1] Historically, the term concept-to-text generation coincided with the term natural language generation (NLG), until more recently other relevant tasks, such as text-to-text generation emerged. Concept-to-text generation systems assume non-linguistic input and thus differ from text-to-text generation applications (e.g., summarisation) which transform text input into some other form (e.g., shorter or simpler) of textual output. In this work we assume the original meaning for NLG, hence we will be using both terms interchangeably.

they will be presented in natural language (*surface realisation*). Another issue is that a lot of manual effort is required to annotate input data, often by consulting experts in a domain; this inevitably increases the cost of the system and constrains its applicability, given the limited number of fully-annotated data available at hand. Therefore the following question naturally arises:

> *How can we achieve cheap and portable concept-to-text generation systems?*

Recent advances in the past ten years in other fundamental fields of NLP such as syntactic parsing, grammar induction, and machine translation (among others), have been all based in statistical models, that draw their knowledge entirely from data. The paradigm shift from rule-based to probabilistic models has started influencing implementation decisions in the generation community as well (Reiter et al., 2005; Belz, 2008; Chen and Mooney, 2008; Angeli et al., 2010; Kim and Mooney, 2010). Data-driven approaches are attractive because we can abstract the internal representation of the input by automatically learning patterns of document planning or surface realisation directly from the data, using mathematically-grounded statistical methods. In this way, we can avoid restrictive heuristics and hand-crafted expert-driven rules. It is possible to port a generation system to different domains and languages merely by re-training it on different input data, while leaving the structure of the model *unchanged*. It is also important to note that the amount of effort required to develop and re-train such methods is considerably less compared to developing rule-based modules; this translates to more economical solutions, which in turn makes their reach to the general public wider.

In this work we will present a set of statistical models that capture the generation process jointly; we assume a simple (widely-accepted) database schema as the representation of the input, and draw all the necessary knowledge to generate the final output text from data. More importantly we show that the same models can port to several domains, via re-training on different input (we experimented on four different domains). Finally, even though we focus on English, the lexicalisation and rendering of text in our models does not make any language specific assumptions.

1.2 Thesis Contributions

The majority of existing end-to-end generation systems (especially the earliest in the field) attribute their success to one or more of the following characteristics:

Date	05/04/13									
Hour (EDT)	14	15	16	17	18	19	20	21	22	23
Temperature (F)	57	57	57	56	54	52	50	50	48	47
Dewpoint (F)	40	38	38	37	37	35	36	36	35	35
Heat Index (F)										
Wind (mph)	16	16	17	17	15	11	9	8	7	6
Wind Dir	E	E	E	E	E	E	E	E	E	E
Gust										
Sky Cover (%)	0	0	0	0	1	3	4	21	38	55
Pcpn. Potential (%)	0	0	0	0	0	0	0	0	0	0
Rel. Humidity (%)	53	49	49	49	52	52	58	58	60	63
Thunder	–	–	–	–	–	–	–	–	–	–
Rain	–	–	–	–	–	–	–	–	–	–

This Afternoon:	Sunny, with a high near 58. East wind around 16 mph.
Tonight:	Increasing clouds, with a low around 40. Northeast wind 6 to 9 mph.

Figure 1.1: Weather forecast example captured from `www.weather.gov`, on 04 May 2013, at 7.28pm for Boston, MA.

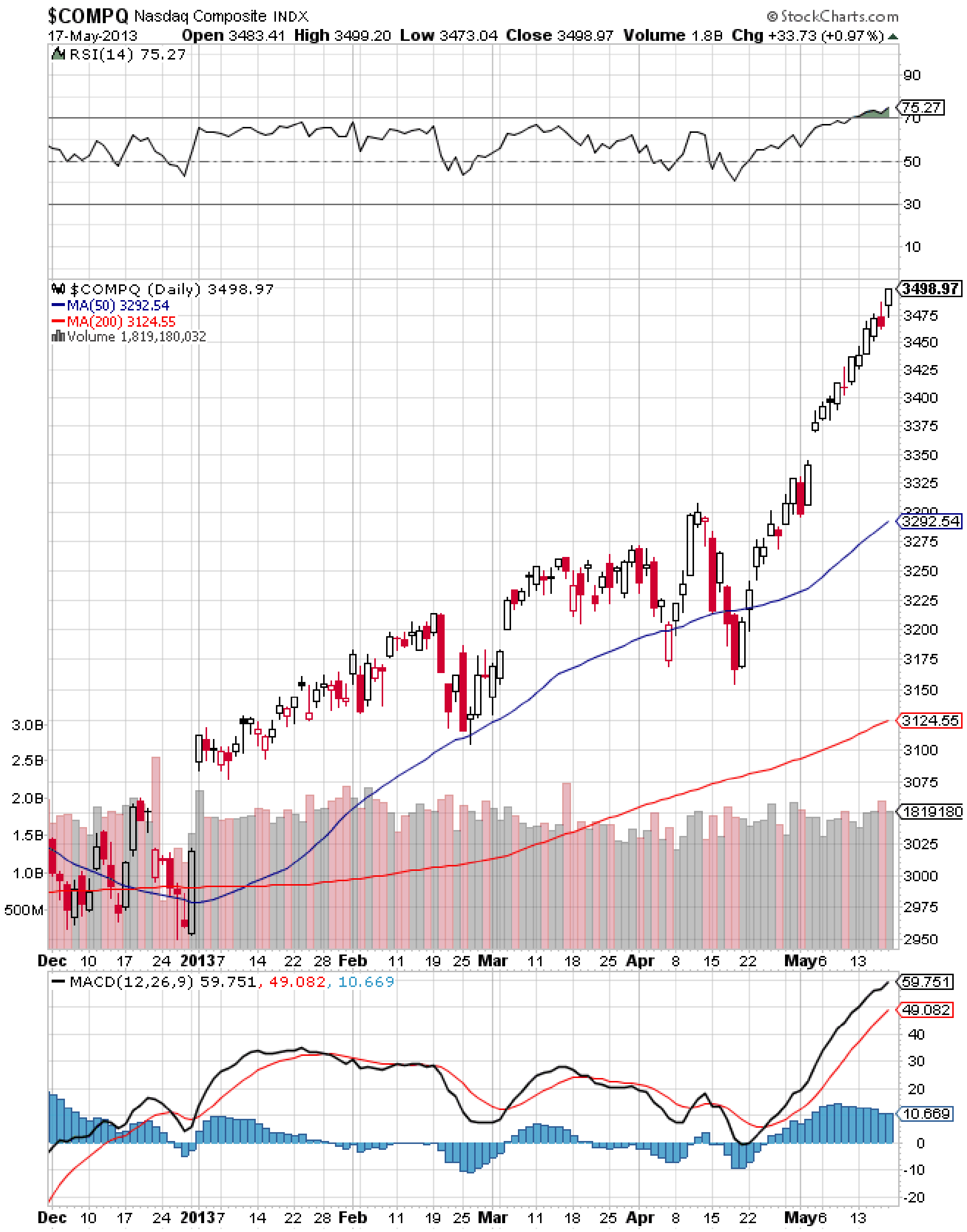
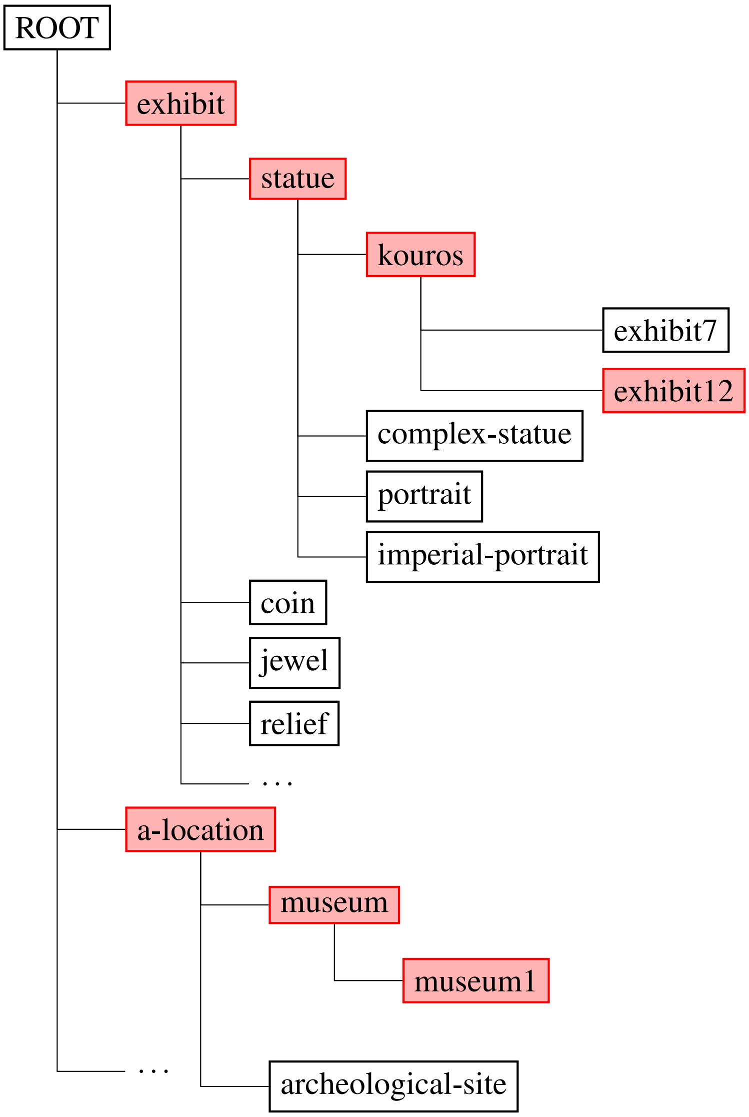


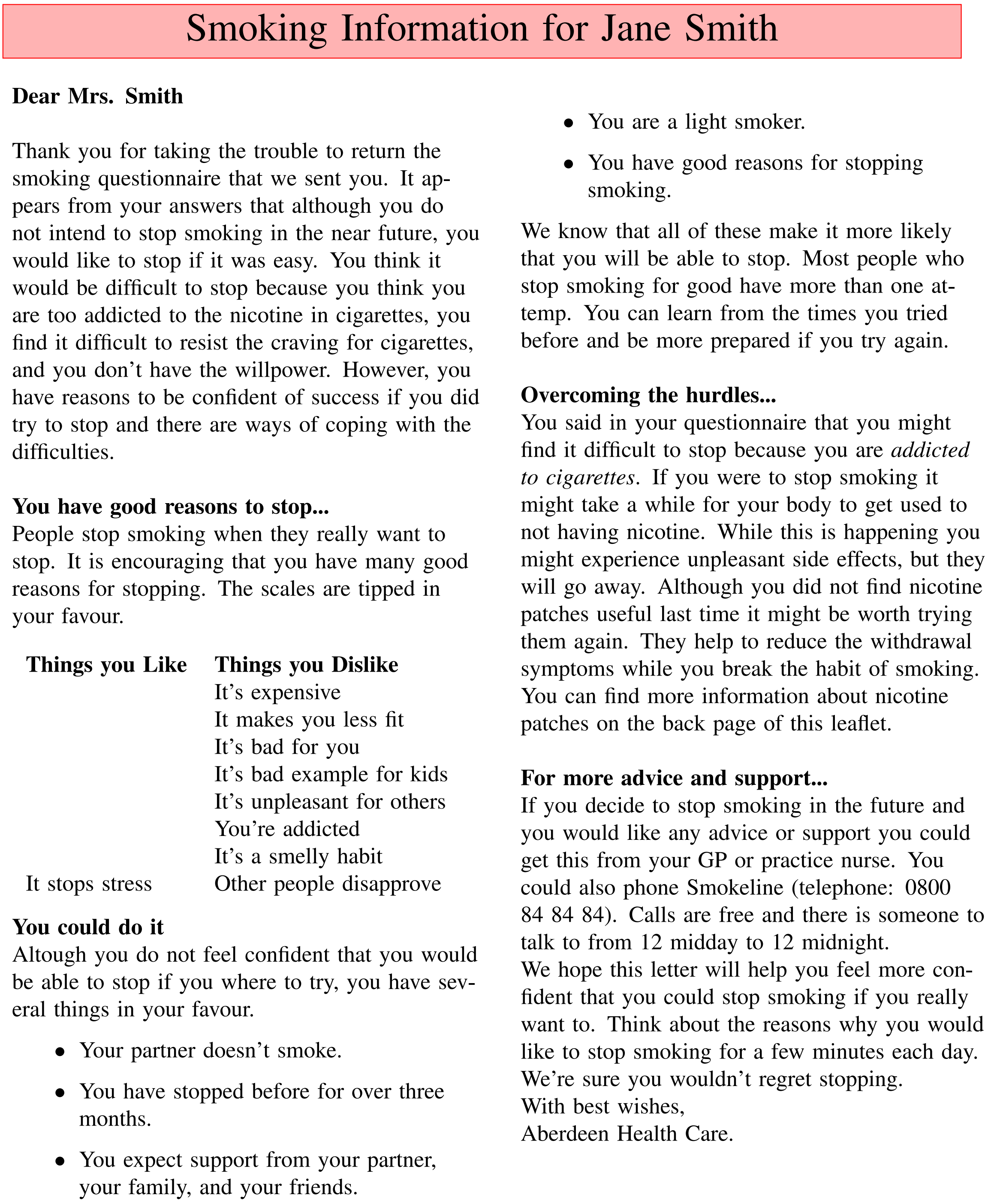
Figure 1.2: Typical stock market trend chart example for the Nasdaq composite index captured from `stockcharts.com`, on 17 May 2013.



This exhibit is a kouros; it was created during the archaic period and it dates from the 6th century B.C. It is made of Parian marble, by Aristion, and it originates from Merenta, Attica. Currently it is in the Archaeological Museum of Athens.

- Other exhibits created during the archaic period:
  - A portrait made from marble
  - A panathenaic amphora originally from Attica
- Other exhibits that originate from Merenta, Attica:
  - A kori created during the archaic period

Figure 1.3: Example input (only a small excerpt is shown) and output of the M-PIRO system (Isard et al., 2003). We highlight in red the part of the input knowledge base hierarchy that is chosen and the corresponding resulting text.



Smoking Information for Jane Smith

Dear Mrs. Smith

Thank you for taking the trouble to return the smoking questionnaire that we sent you. It appears from your answers that although you do not intend to stop smoking in the near future, you would like to stop if it was easy. You think it would be difficult to stop because you think you are too addicted to the nicotine in cigarettes, you find it difficult to resist the craving for cigarettes, and you don't have the willpower. However, you have reasons to be confident of success if you did try to stop and there are ways of coping with the difficulties.

You have good reasons to stop...
People stop smoking when they really want to stop. It is encouraging that you have many good reasons for stopping. The scales are tipped in your favour.

Things you Like	Things you Dislike
	It's expensive
	It makes you less fit
	It's bad for you
	It's bad example for kids
	It's unpleasant for others
	You're addicted
	It's a smelly habit
It stops stress	Other people disapprove

You could do it
Altough you do not feel confident that you would be able to stop if you where to try, you have several things in your favour.

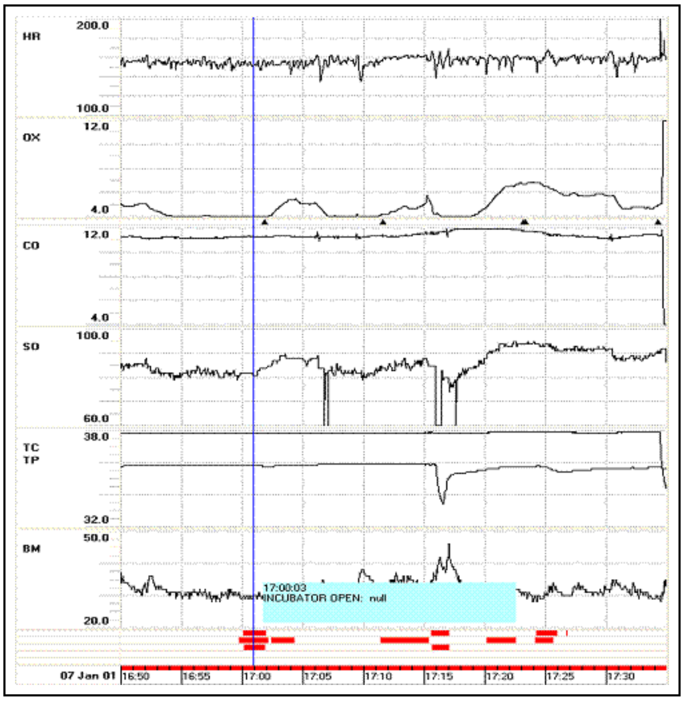
- Your partner doesn't smoke.
- You have stopped before for over three months.
- You expect support from your partner, your family, and your friends.
- You are a light smoker.
- You have good reasons for stopping smoking.

We know that all of these make it more likely that you will be able to stop. Most people who stop smoking for good have more than one attemp. You can learn from the times you tried before and be more prepared if you try again.

Overcoming the hurdles...
You said in your questionnaire that you might find it difficult to stop because you are *addicted to cigarettes*. If you were to stop smoking it might take a while for your body to get used to not having nicotine. While this is happening you might experience unpleasant side effects, but they will go away. Although you did not find nicotine patches useful last time it might be worth trying them again. They help to reduce the withdrawal symptoms while you break the habit of smoking. You can find more information about nicotine patches on the back page of this leaflet.

For more advice and support...
If you decide to stop smoking in the future and you would like any advice or support you could get this from your GP or practice nurse. You could also phone Smokeline (telephone: 0800 84 84 84). Calls are free and there is someone to talk to from 12 midday to 12 midnight.
We hope this letter will help you feel more confident that you could stop smoking if you really want to. Think about the reasons why you would like to stop smoking for a few minutes each day.
We're sure you wouldn't regret stopping.
With best wishes,
Aberdeen Health Care.

Figure 1.4: A letter generated by the STOP system (Reiter et al., 2003).



Sensor Data

Full Descriptor	**Time**
SETTING;VENTILATOR;FiO2 (36%)	10.30
MEDICATION;Morphine	10.44
ACTION;CARE;TURN/ CHANGE POSITION;SUPINE	10.46-10.47
ACTION;RESPIRATION;HANDBAG BABY	10.47-10.51
SETTING;VENTILATOR;FiO2 (60%)	10.47
ACTION;RESPIRATION;INTUBATE	10.51-10.52

Action Records

Doctor Output

By 11:00 the baby had been hand-bagged a number of times causing 2 successive bradycardias. He was successfully reintubated after 2 attempts. The baby was sucked out twice. At 11:02 FIO2 was raised to 79%.

Nurse Output

Respiratory Support - Current Status

Currently, the baby is on CMV in 27% O2. Vent RR is 55 breaths per minute. Pressures are 20/4 cms H2O. Tidal volume is 1.5. SaO2 is variable within the acceptable range and there have been some desaturations.

. . .

Events During the Shift

A blood gas was taken at around 19:45. Parameters were acceptable. pH was 7.18. CO2 was 7.71 kPa. BE was -4.8 mmol/L.

Family Output

John was in intensive care. He was stable during the day and night. Since last week, his weight increased from 860 grams (1 lb 14 oz) to 1113 grams (2 lb 7 oz). He was nursed in an incubator. Yesterday, John was on a ventilator. The mode of ventilation is Bilevel Positive Airway Pressure (BiPAP) Ventilation. This machine helps to provide the support that enables him to breathe more comfortably. Since last week, his inspired Oxygen (FiO2) was lowered from 56% to 21% (which is the same as normal air). This is a positive development for your child.

Figure 1.5: Example input and multiple personalised output texts for different readers generated by the BabyTalk system (Portet et al., 2009).

- Expert knowledge deployed for the creation of hand-crafted rules that define a representation of usually a single domain, or the grammar itself which is also a set of hand-crafted rules. This of course results in high quality text, matching human output.

- The use of manually annotated corpora with features such as discourse relations or alignments between the structural representation of the input and the textual output (McKeown, 1985a; Hovy, 1993).

- The decomposition of the generation process into individual components or modules, so as to alleviate the added engineering effort of performing many steps simultaneously (Goldberg et al., 1994; Reiter et al., 2005). The intermediate input-output flow between modules can benefit from further supervision or rule-based intervention.

More recent approaches attempt to acquire the necessary linguistic knowledge to perform generation directly from data, thus obviating the need for expert rule-based systems. However, they tend to focus on specific components of the generation process, such as content selection (Barzilay and Lapata, 2005a; Snyder and Barzilay, 2007), or surface realisation (Lavoie and Rambow, 1997); rather few data-driven systems exist that tackle generation in an end-to-end manner. Notably, the works of Angeli et al. (2010) and Kim and Mooney (2010) are such examples. However, they rely on some form of human intervention and domain knowledge and still decompose generation into parts.

The work we present in this thesis adopts an entirely data-driven approach, which sets it apart from earlier and more recent work on concept-to-text generation. Our modelling approach is end-to-end: given an input set of database records, we generate an output text that best describes it. More specifically, we propose a set of models that:

- Recast generation as a probabilistic parsing problem. We construct a syntactic probabilistic context-free grammar that globally captures the conceptual input regardless of the domain, and develop several parsing algorithms in order to generate the textual output.

- Learn all the necessary document, structural and linguistic knowledge from unannotated data. During training, we only need some conceptual input in the form of database records and collocated unaligned text.

- Jointly perform all major parts of the generation process, namely ‘what to say’ and ‘how to say’, in a single framework, fully benefiting from their interaction.

Finally, we evaluate our systems on four different real-world domains using both automatic metrics and human judgement studies. Our results are competitive or superior to other data-driven end-to-end systems (Angeli et al., 2010; Kim and Mooney, 2010), without relying on hand-engineering or expert domain knowledge.

1.3 Thesis Outline

The rest of the thesis is structured as follows:

- In Chapter 2 we discuss the consensus pipeline architecture of a typical concept-to-text generation system. Then we provide a brief overview of the field from the early rule-based systems to the more recent data-driven approaches.

- In Chapter 3 we formulate the problem of generating text from a set of database records, where each record has fields and each field takes either a categorical, string or integer value. Then we present the four domains we experimented on, and finally review the evaluation methodology we adopted in order to measure the performance of our systems.

- In Chapter 4 we present our joint concept-to-text generation model. We begin by extending an existing content selection model (Liang et al., 2009), and recasting it into a probabilistic context-free grammar. Then we formulate several decoding algorithms based on the CYK parser (Kasami, 1965; Younger, 1967). We show how to efficiently integrate our decoder with external linguistically motivated models, in order to guarantee fluent and grammatical output. We present an implementation of our decoders using hypergraphs and evaluate it on four domains, achieving results comparable or state-of-the-art against competitive generation models (Angeli et al., 2010; Kim and Mooney, 2010).

- In Chapter 5 we extend the basic model from Chapter 4 by introducing a set of rules that operate on the document level. Our aim is to induce document plans directly from training data, that can better guide the selection and ordering of database records in the final output, based on more global decisions compared to the local content selection of the previous chapter. We also show an efficient way to extract document-level rules from data in order to restrict the exponential search space incurred by the new rules. Experiments on two multi-sentence corpora show that the extended model obtains superior performance compared to the original and the baseline models.

- In Chapter 6 we present an exploratory study on field-level content selection by discriminatively reranking the hypergraph implementation presented in Chapter 4. In this chapter we depart from the unsupervised training of the previous two models (using EM), and experiment with lexical and more importantly field-level features, which we train using the structured perceptron algorithm (Collins, 2002). Evaluation on one challenging domain yields promising results.
- In Chapter 7 we conclude this thesis and discuss some interesting avenues for future research within the realm of data-driven joint models for concept-to-text generation.

Some of the work presented here has been previously published in Konstas and Lapata (2012b, 2013a) (Chapter 4), Konstas and Lapata (2013b) (Chapter 5) and Konstas and Lapata (2012a) (Chapter 6).

Chapter 2

Background

In this chapter we will provide an overview of the field of concept-to-text generation. We begin by describing a consensus pipeline architecture defined by the majority of systems in the 1980s and 1990s, and then focus on some pivotal early generation systems. Then we move on to present more recent data-driven approaches, leading finally to end-to-end probabilistic systems.

2.1 Architecture of a NLG System

2.1.1 Input

According to Reiter and Dale (2000, p. 43) the input to a typical NLG system can be defined as follows:

> we can characterise the input to a single invocation of an NLG system [...] as a four-tuple $\langle k, c, u, d\rangle$, where k is the *knowledge source* to be used, c is the *communicative goal* to be achieved, u is a user model and d is a *discourse history*.

The knowledge source is the domain-specific information available to the system, usually in the form of knowledge base entries, expert system output, ontology structures, database records, or even in the form of formal meaning representations such as lambda calculus. The heterogeneity of applications inevitably leads to varied representations and content; hence there is no actual characterisation of this part of the input, other than it constitutes the entry point of information to the system.

The communicative goal is easier to define as it describes the purpose of the generated text. For example, the communicative goal of a sportscaster NLG system is to generate comments on the events happening at a particular time in the context of

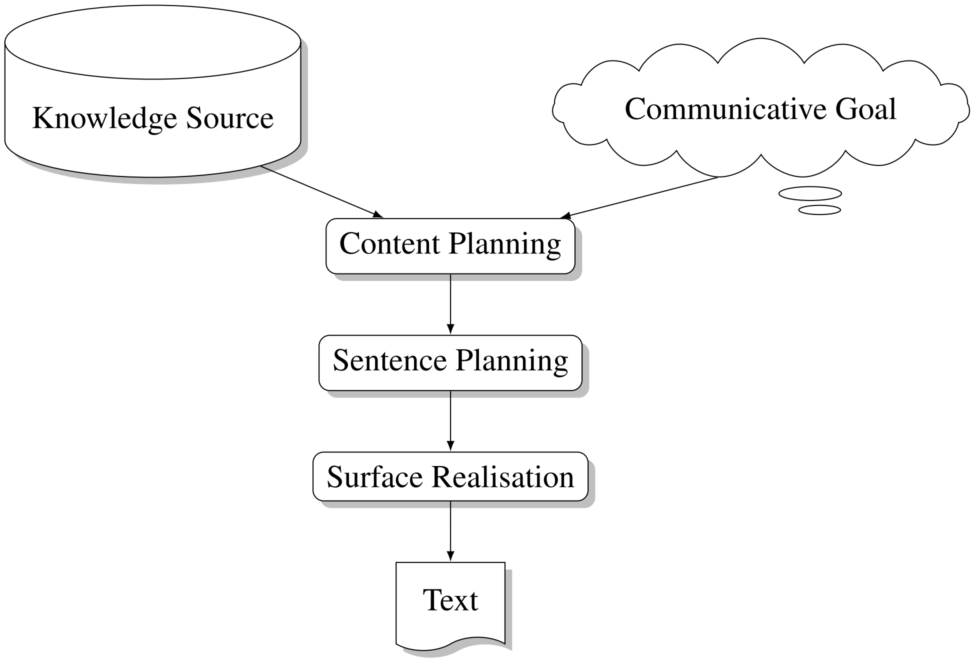


Figure 2.1: Pipeline of modules in a typical NLG system.

a sports game. Similarly, the communicative goal of a weather forecast NLG system (see Figure 1.1) is to output a summary of the weather for the next 24 hours given the current and predicted measurements of various meteorological phenomena.

The user model is a characterisation of the reader for whom the generated text is intended. While usually not explicitly specified in most systems we will examine, there are situations where this parameter can affect the processing steps of a system and its output. For example, a weather forecast generator might produce different texts for different users depending on whether they are experts (e.g., meteorologists or laypersons, farmers, residents in an urban or a coastal area, fishermen, and so on).

Finally, the discourse history models the text that has been generated by the system so far. This is useful in case a system wants to keep track of the entities in the knowledge source that have been mentioned, in order to inform the use, for example, of anaphoric expressions in subsequent references to them, later in the text. This is more common in the NLG part of dialogue systems, which need to keep track of what has been mentioned in each dialogue turn and update their state accordingly. Even in single-interaction systems, i.e., systems which are executed once to generate a text, it might be necessary to aggregate information, which will be reflected in later executions. For example, a weather forecast generator may keep track of previously generated forecasts and produce a sentence contrasting a noteworthy characteristic: *Today it is going to be* **warmer** *than yesterday.* Note, that for most of the systems we will be describing in this chapter, and in the work presented in this thesis, we will assume that the user model and the discourse history are empty, and we will not deal with them henceforth.

2.1.2 Output

The output of a typical NLG system is primarily a *text*. This usually corresponds to a string of words, that form sentences, paragraphs and sections, depending on the domain and application at hand. In addition to text, a system may also output additional information such as prosodic cues, HTML markup, word-processor directives, and so on. The length of the resulting document varies greatly and may range from a single word (e.g., an utterance in a dialogue system) to a multi-paragraph document (see example in Figure 1.4).

2.1.3 Modules Pipeline

We now move on to the actual architecture of a typical generation system. Initial NLG efforts (KAMP; Appelt 1985, Danlos 1984) adopted a rather monolithic approach to generation; there was no clear distinction among (rule-based) decisions that selected, ordered and realised into text the information contained in the knowledge source. Subsequent systems began to form a consensus architecture that breaks down the generation process into a series of modules. Each module is concerned with a specific well-defined task and is separate from the rest. The inter-communication between modules is made possible only via *messages* exchanged between each other; one module takes a particular type of message as input from its predecessor and outputs another type of message to its successor. NLG system designers usually name the modules differently, to accommodate the needs of the problems they each try to solve. In the following we use their most common characterisation (adapted from Reiter (1994) and Reiter and Dale (2000), see Figure 2.1):

Content Planning takes the initial input knowledge source k and produces an internal semantic representation, i.e., an abstract specification of what should be conveyed in the resulting text. This process often entails two separate sub-processes:

Content Selection is the task that determines what parts of the source are going to be chosen to be included in the final text, which parts should be omitted and in what order they should be mentioned.

IDENTIFICATION

(1) Identification (class & attribute/function)

(2) [Analogy/Constituency/Attributive/Renaming]+

(3) Particular-illustration/Evidence+

(4) [Amplification/Analogy/Attributive]

(5) [Particular-illustration/Evidence]

Eltville (Germany) (1) An important wine village of the Rheingau region. (2) The vineyards make wines that are emphatically of the Rheingau style, (3) with a considerable weight for a white wine. (4) Taubenberg, Sonnenberg and Langenstuck are among vine-yards of note.

Figure 2.2: Document planning using the identification schema from TEXT (McKeown, 1985b) along with an example taken from Hovy (1993). Numbers in parentheses indicate the different parts of the schema, brackets indicate optional parts, the forward slash (/) denotes an either-or relation, while the plus symbol (+) denotes one or more instances of the particular part.

Document Planning or document structuring is the part that organises the information emitted by the previous component in a rhetorically coherent manner.

Note that these two sub-modules are often interleaved (Moore and Paris, 1989). In some cases (e.g., in recent data-driven systems) the document planning component can be omitted, for the sake of simplicity (Kim and Mooney, 2010; Angeli et al., 2010).

The output of this module is a *document plan*. A popular representation for a document plan used in early systems are discourse *schemata* (McKeown, 1985b). A discourse schema encodes a predefined set of instructions that define the order in which information from the knowledge source should be mentioned in the resulting test. Figure 2.2 shows the identification schema and an example of it. Another form of representation is trees where the internal nodes denote discourse information and the leaf nodes correspond to parts of the input chosen by the content selection process (Hovy (1993), see Figure 2.3 for an example).

Sentence Planning takes the document plan from the previous module and converts it to linguistic structures, specifying content words and grammatical relationships. In other words, a sentence planner operates as a proxy between the knowledge source

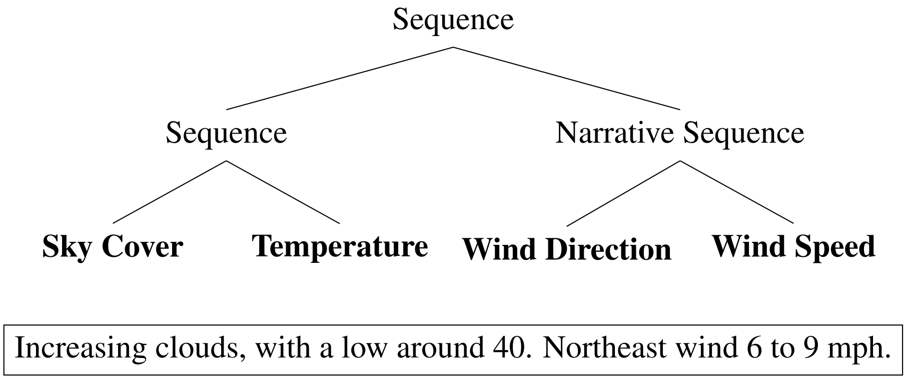


Figure 2.3: Document planning using a tree representation on the 'Tonight' forecast example of Figure 1.1. Internal nodes correspond to RST relationships, while leaf nodes refer to parts of the input.

and the text, and is engaged in decisions such as lexicalisation, generation of referring expressions and aggregation. Briefly, lexicalisation refers to the task of choosing particular words (e.g., from a lexicon) that describe some parts of the input. Generating referring expressions directs the choice of specific words that will enable entities in the input to be identified uniquely and naturally given their context in the text. For example, the referring expression for an entity that corresponds to a name, could be the actual name (e.g. *'Mary'*) if it is mentioned for the first time in the text, and then *'she'* or *'her'* in subsequent mentions. Finally, aggregation is responsible for combining pieces of information in the document plan together at the sentence level. An example in the weather domain could be the conjunction of two semantically relevant entities, as in *'Chances of rain and thunderstorms'* instead of *'Chance of rain. Chance of thunderstorms'*. Figure 2.4 shows an example sentence plan that implements lexicalisation and aggregation.

Similarly to document planning, parts of this module may be omitted or performed jointly. Interestingly, as we will see in sections 2.2–2.3, more recent studies tend to interleave sentence planning with document planning. Finally, the output of the sentence planner is a *text specification*, again usually in the form of a tree, whose nodes describe the structure of the text and whose leaves correspond to sentences. Notable representations include the Sentence Planning Language (Kasper, 1989), Functional Descriptions (Elhadad and Robin, 1998) and Meaning-Text theory (Mel'čuk, 1988).

Surface Realisation is the last module in the pipeline that renders the final text. It takes as input the abstract text specification from the previous module and turns it into

$$\begin{bmatrix}
\text{TYPE: ABSTRACT SYNTAX} & \\
\text{HEAD: } |\text{TO}| & \\
\text{CONJ}_1\text{:} & \begin{bmatrix}
\text{TYPE: ABSTRACT SYNTAX} & \\
\text{HEAD: } |\text{WIND}| & \\
\text{SUBJECT:} & \begin{bmatrix} \text{REFERRING NP} & \\ \text{OBJECT:} & \begin{bmatrix} \text{DIRECTION: NORTHEAST} \end{bmatrix} \end{bmatrix} \\
\text{MODIFIER:} & \begin{bmatrix} \text{REFERRING NP} & \\ \text{OBJECT:} & \begin{bmatrix} \text{SPEED: } 6 \end{bmatrix} \end{bmatrix}
\end{bmatrix} \\
\text{CONJ}_2\text{:} & \begin{bmatrix}
\text{TYPE: ABSTRACT SYNTAX} & \\
\text{HEAD: } |\text{WIND}| & \\
\text{SUBJECT:} & \begin{bmatrix} \text{REFERRING NP} & \\ \text{OBJECT:} & \begin{bmatrix} \text{DIRECTION: NORTHEAST} \end{bmatrix} \end{bmatrix} \\
\text{MODIFIER:} & \begin{bmatrix} \text{REFERRING NP} & \\ \text{OBJECT:} & \begin{bmatrix} \text{SPEED: } 9 \end{bmatrix} \end{bmatrix}
\end{bmatrix}
\end{bmatrix}$$

Figure 2.4: Abstract text specification corresponding to the sentence *'Northeast wind 6 to 9 mph.'* represented in an Attribute-Value Matrix (AVM). ABSTRACT SYNTAX refers to a high-level syntactic description of the resulting surface text, and REFERRING NP corresponds to a complementary noun phrase. Notice the lexicalisation of the direction and speed with the values NORTHEAST and 6 and 9 respectively. Also note the aggregation of the speed values, encoded in the two parts CONJ_1 and CONJ_2. Without aggregation the resulting text would be *'Northeast wind 6 mph. Northeast wind 9 mph.'*

a string of words. Methods to perform this task range from simple canned text or template-based techniques, to rule-based systems based on a formal linguistic theory such as Functional Unification Grammar (SURGE; Elhadad and Robin 1998, FUG; Kay 1984), Lexicalised Tree-Adjoining Grammar (SPUD; Stone and Webber 1998), Meaning-Text theory (REALPRO; Lavoie and Rambow 1997), and Combinatory Categorial Grammar (CCG; White and Baldridge 2003). Figure 2.5 gives an example of an abstract syntactic structure of a sentence, as specified in REALPRO.

The pipeline architecture is mainly motivated by two reasons. From an engineering

$$
\left[\begin{array}{ll}
\text{HEAD: NONE} & \\
\text{TENSE: NONE} & \\
\text{SUBJ:} & \left[\begin{array}{ll}
\text{HEAD: } |\text{WIND}| & \\
\text{NUMBER: SG} & \\
\text{CLASS: COMMON-NOUN} & \\
\text{ATTR:} & \left[\begin{array}{l}\text{HEAD: } |\text{NORTHEAST}| \\ \text{CLASS:ADJECTIVE}\end{array}\right]
\end{array}\right] \\
\text{ATTR:} & \left[\begin{array}{ll}
\text{HEAD: } |\text{MPH}| & \\
\text{COORD:} & \left[\begin{array}{l}
\text{HEAD: } |\text{TO}| \\
\left[\begin{array}{ll}\text{SUBJ:} & \left[\begin{array}{l}\text{HEAD: } |6| \\ \text{CLASS: CARDINAL}\end{array}\right]\end{array}\right] \\
\left[\begin{array}{ll}\text{SUBJ:} & \left[\begin{array}{l}\text{HEAD: } |9| \\ \text{CLASS: CARDINAL}\end{array}\right]\end{array}\right]
\end{array}\right]
\end{array}\right]
\end{array}\right]
$$

Northeast wind 6 to 9 mph.

Figure 2.5: Deep Syntactic Structure example in Attribute-Value (AVM) format, with the corresponding text. This is a typical abstract representation of a sentence in a surface realiser (in this particular case REALPRO (Lavoie and Rambow, 1997)). SUBJ refers to the subject of a clause, ATTR refers to a phrase modifier, and COORD defines a conjunction. Notice how only content words, syntactic roles (e.g., subject of a phrase), and syntactic features (e.g., the number of a noun) are specified; function words and word ordering are decided at the final stage of decoding, subject to surface constraints such as number and tense agreement.

(and hence economical) point of view, it is considered to be easier to build, debug, deploy and maintain a system that consists of many small parts rather than one (Marr, 1976). In a more coupled environment, small changes in one part will potentially reflect changes in many other parts as well; the situation can aggravate when more than one developer is involved. From a cognitive point of view, there is some evidence that supports the modular structure. In particular, clinical studies conducted by Ellis and

Young (1996) with brain damage patients, showed that subjects who had lost certain abilities did not suffer overall degradation in all the rest of their abilities. This claim empirically supports the possibility that particular cognitive processes are taking place in partial isolation from the rest. For example, they showed that patients could produce syntactically correct utterances but could not organize utterances into coherent wholes, or analogously they could perform surface generation but not content planning.

While the modular decomposition described above has been generally adopted, a few approaches have relaxed the strict independence assumption, by allowing some local feedback between adjacent modules (Rambow and Korelsky, 1992; Stone and Doran, 1997). In general, the main argument against the strict pipeline architecture stems from the fact that some linguistic phenomena can be better explained when looking at constraints from different levels, such as morphology, syntax and semantics, at the same time and compositionally, rather than in isolation. Another reason for the popularity of the modular architecture is mostly historic. Computing resources are limited, software engineering paradigms such as object-oriented programming, are yet to be invented, or are used in limited scale, hence the need to adopt pipeline techniques.

In the following section we will focus on some well-known pipeline-based systems focusing first on module implementations and then on complete systems. Then we will describe more recent systems that adopt some notion of probabilistic modelling and interestingly, depart from the idea of strict modularity.

2.2 Pipeline-based Systems

The literature reveals many examples of generation systems that produce high quality text, almost indistinguishable from human writing (Dale et al., 2003; Reiter et al., 2005; Green, 2006; Turner et al., 2009; Galanis and Androutsopoulos, 2007). Such systems often involve a great deal of manual effort. For instance, a typical content planning 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. Sentence planning usually calls for engineering templates that capture stereotypical paragraph structures. Analogously, surface realization is often based on a grammar written by hand so as to cover the syntactic constructs and vocabulary of the domain. Finally, both the input and the inter-communication messages require some form of supervision, such as discourse-level annotation of the input text or rule-based categorisation of the document plans in order to identify cues for aggregation.

2.2.1 Individual Components

The work of Hovy (1993) is the most notable early study on content planning based on rhetorical structure theory, (Mann and Thompson, 1988)[1]. The RST relations are encoded as operations in the context of an AI planner. They are recursively combined, in order to satisfy as many entities of the input knowledge source as necessary and are subject to manually created coherence restrictions. The output is a document plan tree with RST relations as intermediate nodes, and nuclei and satellite content words—manually mapped to the input entities—as leaves.

SPUD (Sentence Planner Using Descriptions, Stone and Doran 1997) is a sentence planner that integrates constraints from syntax, semantics and pragmatics into a common framework using Lexicalised Tree-Adjoining Grammar (LTAG; Joshi 1987). The system assumes a sequence of goals from a content planner, and incrementally combines elementary trees augmented with domain-specific semantic knowledge, using several predefined rhetoric conditions that guide the search in a top-down fashion. Since the elementary trees are lexicalised, SPUD also performs realisation simultaneously during the parsing process.

REALPRO (Lavoie and Rambow, 1997) is a surface realiser which accepts as input a text specification in Deep-Syntactic Structure representation (Mel'čuk, 1988)[2]. Using a hand-crafted wide coverage grammar of the target language, and a lexicon of domain-specific lexemes provided a priori by the sentence planner, it performs a series of processes such as function word insertion, linearisation of the tree representation, morphological inflection, and surface formatting (e.g., adding punctuation).

Finally, an interesting line of research on surface realisation from logical forms is using chart realisation algorithms (Shieber, 1988; Kay, 1996; Moore, 2002; Carroll et al., 1999; White and Baldridge, 2003; White, 2004). The underlying idea is that we can use the same grammar for both parsing and generation; instead of transducing strings to logical forms, we transduce logical forms to strings. We achieve that by keeping track of partial string derivations in an agenda-based data structure and combining them in a particular order according to the grammar rules and parsing strategy. Shieber (1988) outlines an algorithm for chart realisation using instances of

[1] Briefly, RST defines relationships between two non-overlapping text spans, namely the *nucleus* and the *satellite*. A relation definition consists of constraints on the nucleus, constraints on the satellite, constraints on the combination of the nucleus and the satellite, and of the effect. Example relations are *elaboration, exemplification, contrast and narrative sequence*.

[2] In Meaning-Text theory information is represented as a lexicalised, labelled (i.e., using syntactic descriptors such as *subject*) dependency tree. The nodes of the tree are meaning-bearing lexemes (i.e., not function words).

the architecture for Earley and shift/reduce parsing. Similarly, Kay (1996) introduces an algorithm schema for surface realising input in conjunctive logical form such as that of Davison (Davidson, 1980). Moore (2002) proposes an efficient chart realiser for unification grammars that is based on bottom-up parsing strategy, and guarantees polynomial time under modest constraints on the expressivity of the semantic input. White and Baldridge (2003) implement surface realisation using CCG and an efficient bottom-up parser, addressing phenomena including argument cluster coordination, and gapping.

2.2.2 End-to-End Systems

One of the earliest commercially successful systems that exemplifies the pipeline approach is FOG (Goldberg et al., 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 for content planning 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).

TEXT (McKeown, 1985b) uses a database from the Office of Naval Research and provides information about vehicles and destructive devices. It combines content planning and sentence planning in one module called *'strategic generation'*, by making use of manually crafted schemata of discourse, based on the rhetoric predicates of Grimes (1975). After the necessary information from the knowledge source is selected, it is represented as an ATN and then realised into natural language using a hand-written grammar and dictionary.

More recently, Reiter et al. (2005) 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 et al., 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.

2.3 Data-driven Systems

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.

2.3.1 Individual Components

Barzilay and Lapata (2005b) 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 (still focusing only on content selection), 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. We will describe this model in more detail in Section 4.1.

Duboue and McKeown (2001) present perhaps the first empirical approach to content planning. They use techniques from computational biology to learn the basic patterns contained within a plan and the ordering among them. Duboue and McKeown (2002) learn a tree-like planner from an aligned corpus of semantic inputs and corresponding human-authored outputs using evolutionary algorithms.

Barzilay and Lapata (2006) formulate the sentence planning sub-task of aggregation at the conceptual level, as a supervised set partitioning problem where each partition corresponds to a sentence. Given an aligned corpus of database entries and corresponding sentences, they build a model that partitions a set of input entities into non-overlapping subsets, subject to local (pairwise similarities) and global (overall length) constraints. They encode their model as an integer linear program (ILP) and solve it using standard optimisation tools.

Several data-driven approaches focus on (some parts of) sentence planning and surface realisation in a common modelling framework. We briefly mention here systems that specifically map meaning representations (e.g., some logical form or numeric weather data) to natural language, using explicitly aligned sentence/meaning pairs as training data. WASP^{-1} (Wong and Mooney, 2007) learns 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 Earley chart generator. Knight and Hatzivassiloglou (1995) create word lattices from an existing sentence plan using a syntactic grammar and hand-crafted heuristics, and then perform Viterbi search on the former by using n-gram language models in order to produce the surface string. 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 manually crafted 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 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.

2.3.2 End-to-End Systems

A few approaches have emerged more recently that combine content selection and surface realization. Kim and Mooney (2010) present a generator that produces sportscasting comments of robotic football games. They adopt a two-stage pipeline architecture: using a generative model similar to Liang et al. (2009), they first perform content selection in order to select the salient entities from the knowledge source and then verbalize the selected input with WASP^{-1} (Wong and Mooney, 2007), described above.

In contrast, Angeli et al. (2010) propose a unified content selection and surface realization model which also operates over the alignment output produced by Liang et al. (2009). 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.

In the following we will also present an end-to-end system that performs content planning, rudimentary sentence planning (i.e., lexicalisation) 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 it 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 a 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. The approach is conceptually simple, does not rely on labelled data or any feature engineering. In order to generate fluent output we intersect our grammar with external linguistically motivated models, and search approximately both for the best derivation tree and generated string. Our generator is not strictly modular; rather it models the different components of the generation process jointly, thus allowing them to communicate naturally and influence each other.

2.4 Summary

To summarise, in this chapter we presented the consensus architecture for traditional concept-to-text generation systems. The latter consists of three modules, namely *content planning*, *sentence planning*, and *surface realisation*. We reviewed some early notable work on generators that tackle each module individually, as well as some prominent pipeline-based end-to-end systems. We then presented more recent data-driven approaches that model each module in isolation, and concluded with a series of empirical end-to-end systems. In the following chapter we will define our generation task, the input to our model, and the architecture of our system. We will also describe the datasets we used, as well as our experimental methodology.

Chapter 3

Task Definition

In this chapter we define our generation task. We describe the input our model assumes and give a general overview of the architecture of our system. Then we describe the datasets we used, and finally outline the automatic evaluation metrics we use, as well as the methodology we follow with regard to human evaluation.

3.1 Problem Formulation

Each record token $r_i \in \mathbf{d}$, with $1 \leq i \leq |\mathbf{d}|$, has a type $r_i.t$ and a set of fields $\mathbf{f}$ associated with it. Fields have different values $f_k.v$ and types $f_k.t$ (i.e., integer, categorical and string), with $1 \leq k \leq |\mathbf{f}|$. For example, Figure 3.1 shows two records of type **Wind Speed**, with four fields: *time, min, mean*, and *max*. The values of these fields are `06:00-21:00`, `15`, `20`, and `30`, respectively for the first record and `21:00-04:00`, `0`, `5`, and `8` for the second record; the type of *time* is categorical, whereas all other fields are integers.

The training corpus consists of several *scenarios*, i.e., database records[1] **d** paired with texts w like those shown in Figure 3.1. 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,

[1] We do not make any specific assumptions as far as the database schema, or the format according to which the records are stored (e.g., relational database, csv flat files, etc.), are concerned. We implemented several converters from popular relational database management systems (DBMS) such as MySQL, and from the proprietary formats of each input domain we experimented on. The converted input to our systems is in a flat, human-readable format as shown in Appendix A.

Wind Speed

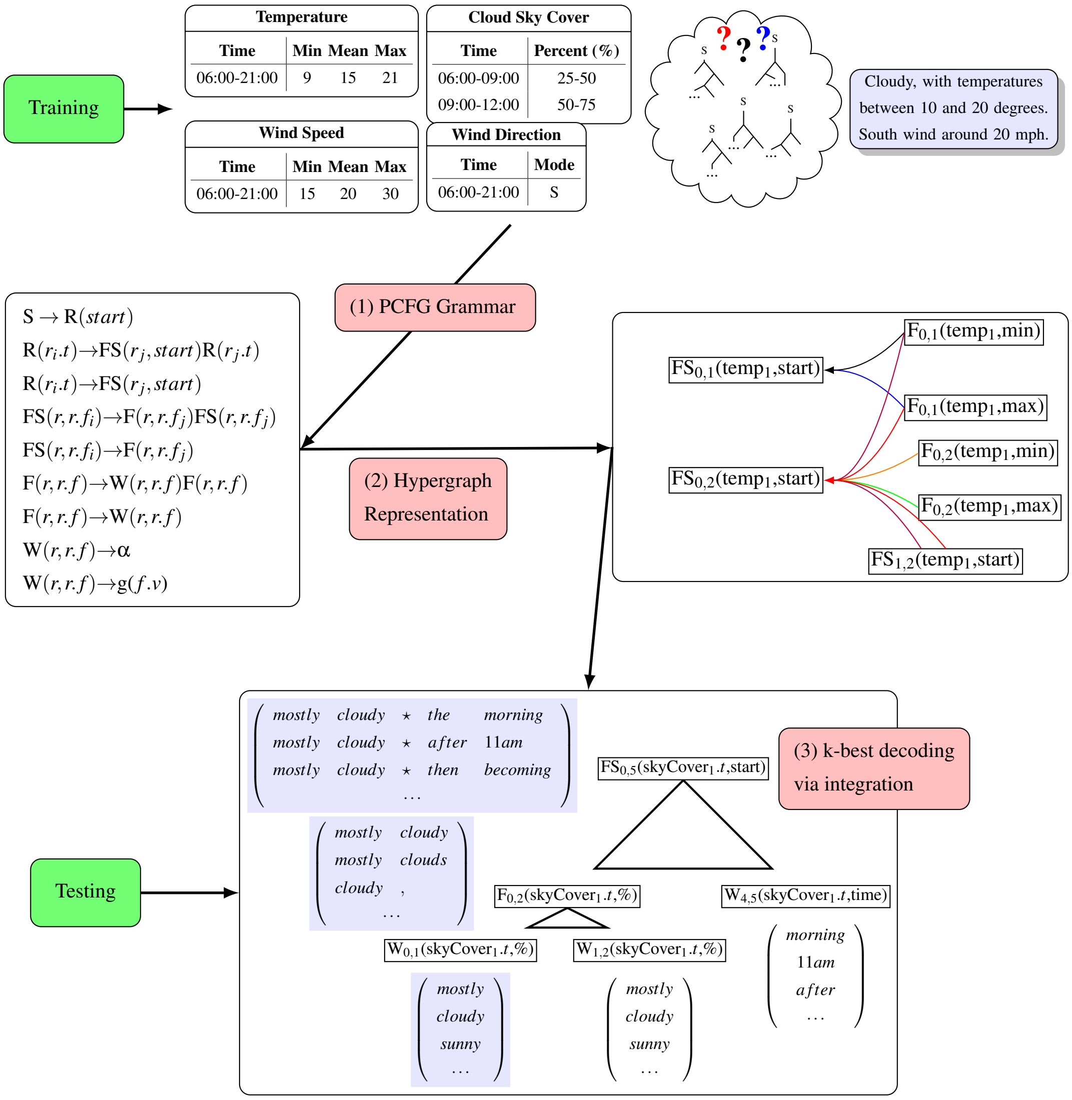
Time	Min (%)	Mean (%)	Max (%)
06:00-21:00	15	20	30
21:00-04:00	0	5	8

between 15 and 30 mph.

Figure 3.1: Two database records from the WEATHERGOV domain, of type **Wind Speed** with the corresponding text '*between 15 and 30 mph.*'. The record in red is the corresponding record to the text. Note that this correspondence is not known during training.

day, time). For the troubleshooting guide, a scenario is a series of user interface (UI) actions to perform on an operating system's desktop environment (e.g., left-clicking on icons, typing into text fields) and a small document describing this process in detail. Note that none of these corpora is parallel, i.e., we do not use any form of alignment between the database and the text. By relaxing this restriction we are faced with a greater challenge as we need to learn the alignments as part of our model (see sections 3.3.1 and 4.2). However, it is a more realistic approach, given that most of the readily available domains, for example on the WWW, such as product specifications and their reviews, statistics of a football game and the summary of it, and so on, do not contain any form of annotation or co-ordination between the text and the corresponding database. During testing, we assume our generator takes as input a set of database records **d** and outputs a text g that verbalizes some of these records.

In Figure 3.2 we outline our system architecture. 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 Section 4.1. For now suffice it to say that our key idea is to recast this model as a probabilistic context-free grammar (PCFG), therefore reducing the tasks of content selection and surface realization into a common parsing problem (Section 4.2). An alternative would be to learn a SCFG 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



Temperature

Time	Min	Mean	Max
06:00-21:00	9	15	21

Cloud Sky Cover

Time	Percent (%)
06:00-09:00	25-50
09:00-12:00	50-75

Wind Speed

Time	Min	Mean	Max
06:00-21:00	15	20	30

Wind Direction

Time	Mode
06:00-21:00	S

$S \to R(start)$
$R(r_i.t) \to FS(r_j, start) R(r_j.t)$
$R(r_i.t) \to FS(r_j, start)$
$FS(r, r.f_i) \to F(r, r.f_j) FS(r, r.f_j)$
$FS(r, r.f_i) \to F(r, r.f_j)$
$F(r, r.f) \to W(r, r.f) F(r, r.f)$
$F(r, r.f) \to W(r, r.f)$
$W(r, r.f) \to \alpha$
$W(r, r.f) \to g(f.v)$

Training
Cloudy, with temperatures between 10 and 20 degrees. South wind around 20 mph.
s
?
?
?
(1) PCFG Grammar
(2) Hypergraph Representation
$F_{0,1}$(temp$_1$,min)
$FS_{0,1}$(temp$_1$,start)
$F_{0,1}$(temp$_1$,max)
$F_{0,2}$(temp$_1$,min)
$FS_{0,2}$(temp$_1$,start)
$F_{0,2}$(temp$_1$,max)
$FS_{1,2}$(temp$_1$,start)
Testing
mostly cloudy ⋆ the morning
mostly cloudy ⋆ after 11am
mostly cloudy ⋆ then becoming
...
mostly cloudy
mostly clouds
cloudy ,
...
$FS_{0,5}$(skyCover$_1$.t,start)
(3) k-best decoding via integration
$F_{0,2}$(skyCover$_1$.t,%)
$W_{4,5}$(skyCover$_1$.t,time)
$W_{0,1}$(skyCover$_1$.t,%)
$W_{1,2}$(skyCover$_1$.t,%)
morning
11am
after
...
mostly
cloudy
sunny
...
mostly
cloudy
sunny
...

Figure 3.2: Outline of our system architecture. During training we need to find the hidden correspondence between database records **d** and text w. We achieve this by recasting this problem into a PCFG grammar which we represent using hypergraphs (1). We train the weights of the grammar directly on the hypergraph (2). During testing we output text g given only the database records **d**, by performing k-best decoding via integration with linguistically motivated models (3). In pink we show the three different stages of our model, and in light blue the best derivation path chosen by the decoder in the last phase.

corpus), as well as grammar training and decoding using state-of-the art SMT methods, namely log-linear training, incorporating length penalisation, defining the beam of the search, which we manage to avoid with our simpler approach. The conceptualization of the generation task 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 (Section 4.3.1). 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 (Section 4.3.2). Thus, our model will generate the text deemed most likely by the grammar *and* the surface models.

We represent the grammar using weighted directed hypergraphs (Gallo et al., 1993). During training, for each input scenario we create a hypergraph following the procedure of Klein and Manning (2001); the weights on the hyperarcs correspond to the weights of the rules of the PCFG. We estimate them using the EM algorithm and a dynamic program similar to the inside-outside algorithm (Li and Eisner, 2009). During testing, given only the set of database records $\mathbf{d}$, we generate text output g, by building a hypergraph and then run a dynamic program equivalent to Viterbi that searches for the best scoring path. While searching, we intersect with the surface level models and create k-best lists of derivation paths in the hypergraph, thus optimizing what to say and how to say at the same time. We describe the representation of the grammar using the hypergraph framework in detail in Section 4.3.3.

3.2 Datasets

We used our system to generate soccer commentaries, weather forecasts, spontaneous utterances relevant to the air travel domain and troubleshooting guides for an operating system. Our aim is to assess how our approach performs under databases of varying size and vocabulary. The four domains cover different registers; they range from written text to spoken dialogue and instructional manuals. They also differ with respect to the size of generated documents, ranging from one sentence to several. Table 3.1 presents corpus statistics across all four domains employed in this thesis. In the following we describe each dataset individually. Each corpus is broken down into a training set, a development set and a test set, unless noted otherwise. Each scenario consists of a database chunk and its corresponding text; we also obtain either manually or automatically the alignments between the records and the parts of the text that they refer to (see Sections 3.2.1-3.2.4). The latter are only used for evaluation purposes during the

Dataset	docs	sents	sents/doc	$\lvert$doc$\rvert$	$\lvert$sent$\rvert$	1-grams	$\lvert r.t \rvert$	$\lvert r_i \rvert$	$\lvert r_i \rvert$/doc
RoboCup	1,539	1,539	1	5.7	5.7	244	9	2.41	2.4
WeatherGov	29,528	81,337	3.25	29.3	9.29	345	12	36	5.8
Atis	4,962	4,962	1	11.2	11.2	927	19	3.79	3.79
WinHelp	128	432	4.3	51.92	11.91	629	13	9.2	9.2

Table 3.1: Corpus statistics for RoboCup, WeatherGov, Atis and WinHelp. The columns (starting from the second), correspond to the total number of documents (docs), total number of sentences (sents), average number of sentences per document (sents/docs), average number of words per document ($\lvert$doc$\rvert$), average number of words per sentence ($\lvert$sent$\rvert$), number of unique words or unigrams (1-grams), average number of record types ($\lvert r.t \rvert$), average number of records per scenario ($\lvert r_i \rvert$), and average number of record alignments per scenario ($\lvert r_i \rvert$/doc), respectively.

training of our models. We give examples of scenarios along with their alignments for each domain in Appendix A.

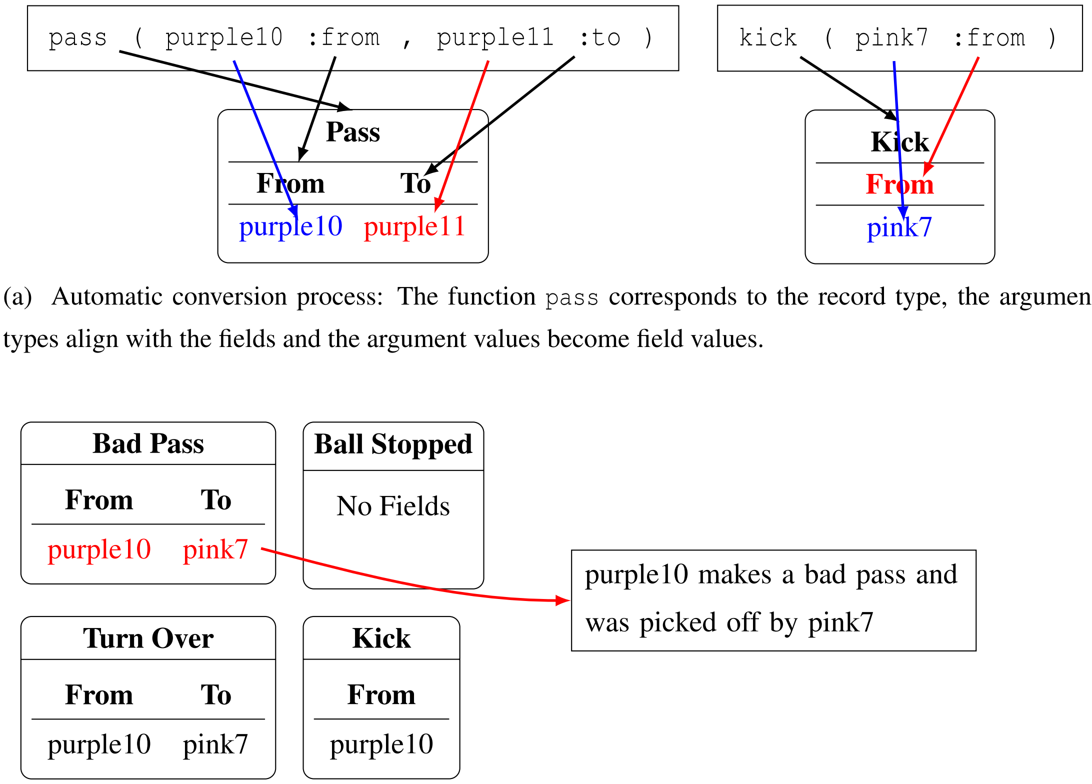
3.2.1 RoboCup Dataset

A RoboCup game is a soccer match between two teams of autonomous robots, such as the NAO humanoid robots (Gouaillier et al., 2008). We used the corpus of Chen and Mooney (2008) which contains manually edited transcriptions of the commentaries from the 2001–2004 RoboCup game finals (henceforth RoboCup). These were created by the commentators during the course of a game. The corpus also contains a semantic representation for some of the sportscasting comments. Chen and Mooney (2008) developed a symbolic representation of game events, most of which involve actions with the ball, such as kicking and passing. The events were automatically extracted from the game logs using a rule-based system, and were represented as atomic formulas in predicate logic. The lexicon and context-free rules for parsing the logical formulas were manually crafted for the specific domain.

We automatically converted the logical formulas into the schema of database records described in the previous section. The process is as follows (see example 3.3a for an illustration)[2]:

- The function name becomes the record type,

[2] The resulting dataset is available from `http://homepages.inf.ed.ac.uk/s0793019/index.php?page=resources`



(a) Automatic conversion process: The function `pass` corresponds to the record type, the argument types align with the fields and the argument values become field values.

(b) Example scenario. Notice that only one record aligns with the corresponding sentence.

Figure 3.3: ROBOCUP: Automatic conversion process from formal language representation to database schema (a), and example scenario (b).

- the argument types of the function correspond to the fields of the record type (the maximum arity in the dataset is 2),

- and the argument values are assigned to the field values.

Each scenario consists of a single line of commentary and the accompanying events, which were extracted automatically from the original logs based on the following heuristic: let the events occuring within a time-window of 5 seconds of the timestamp of the comment, be the possible candidates. Note that only one of those actually corresponds to the comment line. This process generated the final dataset, which consists of 1,539 scenarios containing database records with the corresponding semantic representation and the supporting commentaries. Figure 3.3b shows an example scenario.

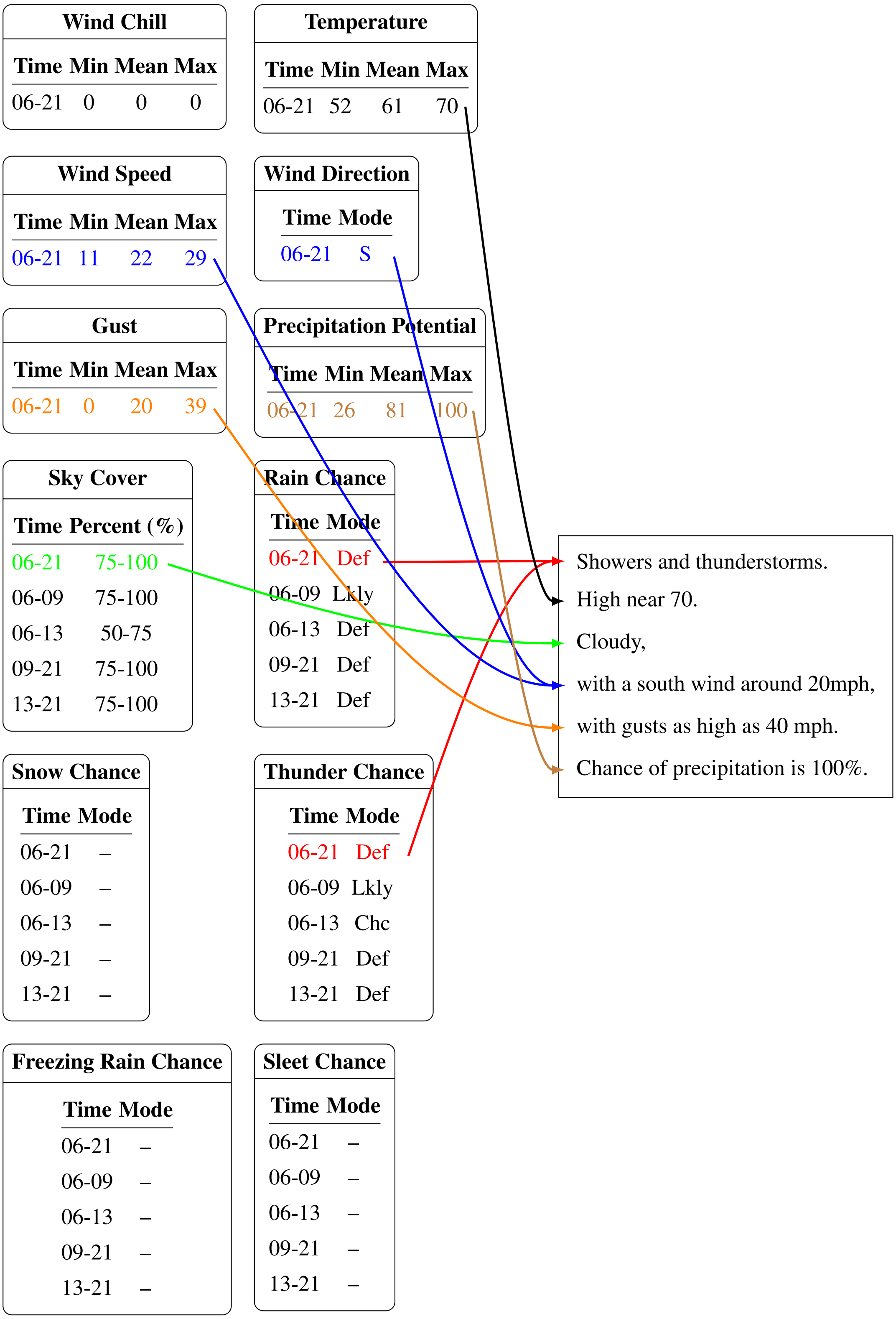
Each scenario in this dataset contains on average $|\mathbf{d}| = 2.4$ records, with categorical values only and is paired with a short sentence (5.7 words). This domain has a small vocabulary (244 words) and simple syntax (e.g., a transitive verb with its subject and object). There is a single record for each scenario that matches the whole sentence of the corresponding comment. Therefore the task of our generator reduces to selecting the correct record out of the set of candidate records present in the same 5-second time-window, hence the content selection is rather trivial. Then the generator lexicalises it by performing surface realisation. Note that the gold-standard records in this dataset were manually aligned to their corresponding sentences (Chen and Mooney, 2008). Given the relatively small size of this dataset, we performed cross-validation following previous work (Chen and 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.

3.2.2 WEATHERGOV Dataset

The second domain we deal with is weather forecasts; we used the dataset of Liang et al. (2009). Each scenario contains a set of database records specific to the local weather forecast and a short text with the accompanying weather report. The authors collected local forecasts for 3,753 major US cities, with a population of at least 10,000, over three days (7–9 February 2009) from `www.weather.gov`. For each city and date, they created two scenarios, corresponding to the day (06:00-21:00) and night (17:00-06:00(+1 day)) forecast, respectively[3]. The original forecasts contain hour-by-hour predictions of measurements of temperature, wind speed, wind temperature, sky cover, rain chance and so on. In order to avoid sparsity issues, the authors aggregated measurements over specific time intervals, e.g. the minimum, maximum and mean wind speed from 06:00 to 21:00.

The resulting dataset consists of 29,528 weather scenarios; the vocabulary in this domain (henceforth WEATHERGOV) is comparable to ROBOCUP (345 words), however, the texts are longer, with 29.3 words per document, and are more varied. On average, each forecast has 3.25 sentences and the content selection problem is more challenging; only 5.8 out of the 36 records per scenario (with 12 record types in total) are mentioned in the text which roughly corresponds to 1.4 records per sentence. The fields of the record types are either of categorical or integer type. Finally, the authors annotated the data at the record level automatically using hand-crafted heuristics, in order to identify which records match which parts of the text. To achieve that, they split the text by punctuation into lines and labelled each line with the records the line

[3] The resulting dataset is available from `http://cs.stanford.edu/~pliang/data/weather-data.zip`



Wind Chill

Time	Min	Mean	Max
06-21	0	0	0

Temperature

Time	Min	Mean	Max
06-21	52	61	70

Wind Speed

Time	Min	Mean	Max
06-21	11	22	29

Wind Direction

Time	Mode
06-21	S

Gust

Time	Min	Mean	Max
06-21	0	20	39

Precipitation Potential

Time	Min	Mean	Max
06-21	26	81	100

Sky Cover

Time	Percent (%)
06-21	75-100
06-09	75-100
06-13	50-75
09-21	75-100
13-21	75-100

Rain Chance

Time	Mode
06-21	Def
06-09	Lkly
06-13	Def
09-21	Def
13-21	Def

Snow Chance

Time	Mode
06-21	–
06-09	–
06-13	–
09-21	–
13-21	–

Thunder Chance

Time	Mode
06-21	Def
06-09	Lkly
06-13	Chc
09-21	Def
13-21	Def

Freezing Rain Chance

Time	Mode
06-21	–
06-09	–
06-13	–
09-21	–
13-21	–

Sleet Chance

Time	Mode
06-21	–
06-09	–
06-13	–
09-21	–
13-21	–

Showers and thunderstorms.
High near 70.
Cloudy,
with a south wind around 20mph,
with gusts as high as 40 mph.
Chance of precipitation is 100%.

Figure 3.4: WEATHERGOV example scenario with automatically extracted gold standard alignments. Notice that a record type may have many records, as in the case of **Sky Cover**, **Rain Chance**, etc.

refers to. The heuristics performed mostly anchor matching between database records and words in the text (e.g., the value `Lkly` of the record **Rain Chance**, matches with the string *'rain likely'* in the text). An example scenario is given in Figure 3.4. In our experiments 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 Angeli et al. (2010).

3.2.3 ATIS Dataset

For the air travel domain we created a corpus based on the ATIS dataset (Dahl et al., 1994). The original corpus contains a total of 5,426 transcriptions of spontaneous utterances of users interacting with a hypothetical online flight booking system, along with the corresponding SQL queries associated with their interaction/requests. Instead of converting this version of the corpus to our database schema, we used the dataset introduced in Zettlemoyer and Collins (2007). The reason is that the original corpus contains user utterances of single dialogue turns which would result in trivial scenarios. Instead 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. The scenarios in the latter dataset consist of the new concatenated utterances and their formal meaning representation in lambda calculus, which we converted to the database scheme as follows (see Figure 3.5 for an illustration of the process)[4]:

1. First we identify records, one per variable (e.g., x); the record type is set according to the type of the variable. We usually determine this from functions with one argument, which primarily play the role of formally assigning a type to the variable. In the example x is of type flight, since function `flight(x)` takes arguments of type flight, $x : fl$. Variables are introduced by the following expressions: `lambda`, `exists`, `argmax`, `argmin`, `min`, `max`, `sum`, `the` and `count`.

2. Then we identify one or more special **Search** record types. These automatically get assigned a *type* field which takes as a value one of the above expressions (e.g., `argmin`, `max`, `lambda` becomes `query`, etc.). They also get a field *what*, which is a reference to the record type they apply to. In the example, the value

[4] The resulting dataset is available from `http://homepages.inf.ed.ac.uk/s0793019/index.php?page=resources`

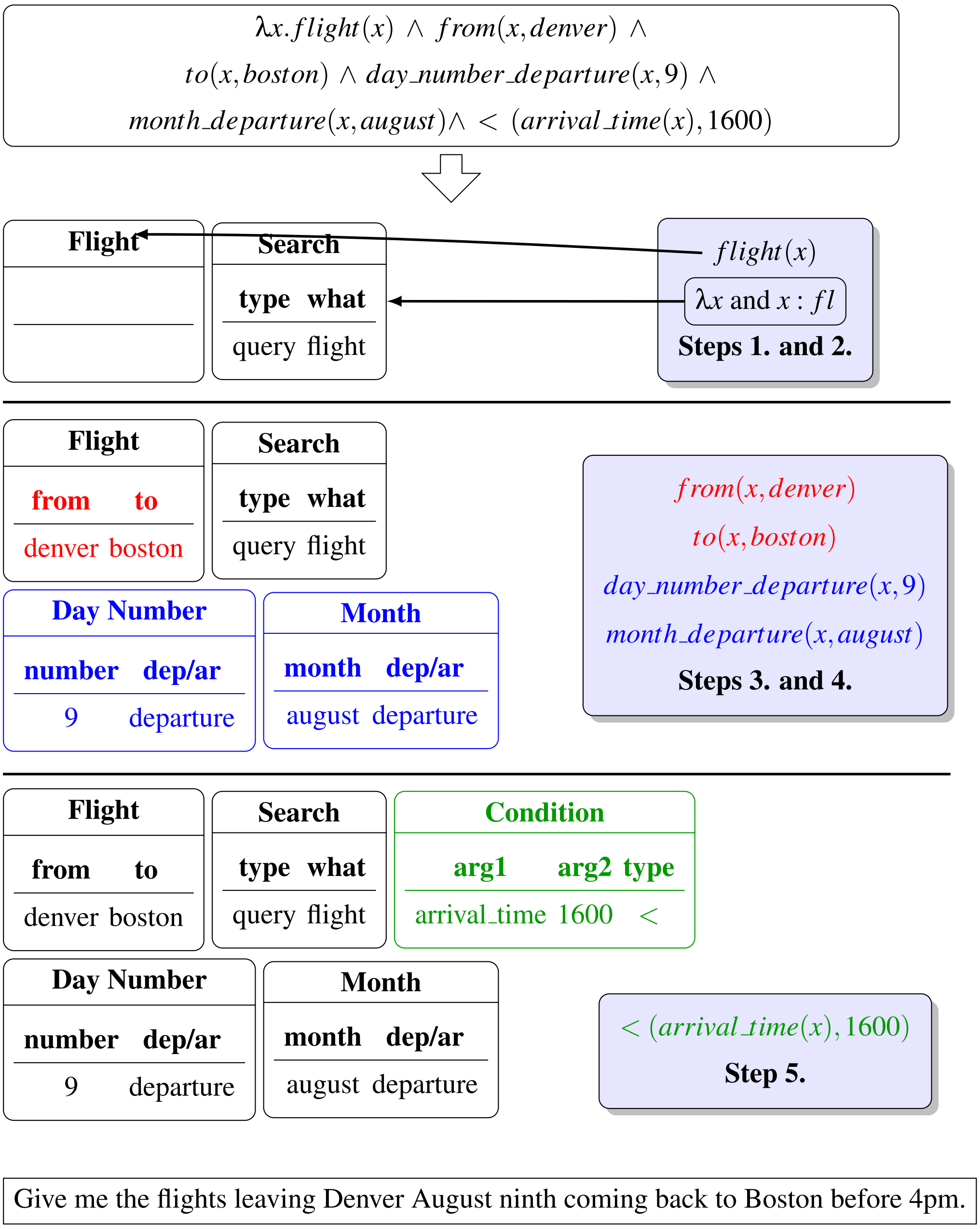


Figure 3.5: ATIS: Automatic conversion process from lambda calculus expressions to the database schema. For a detailed description of how this is achieved we refer the reader to the main text.

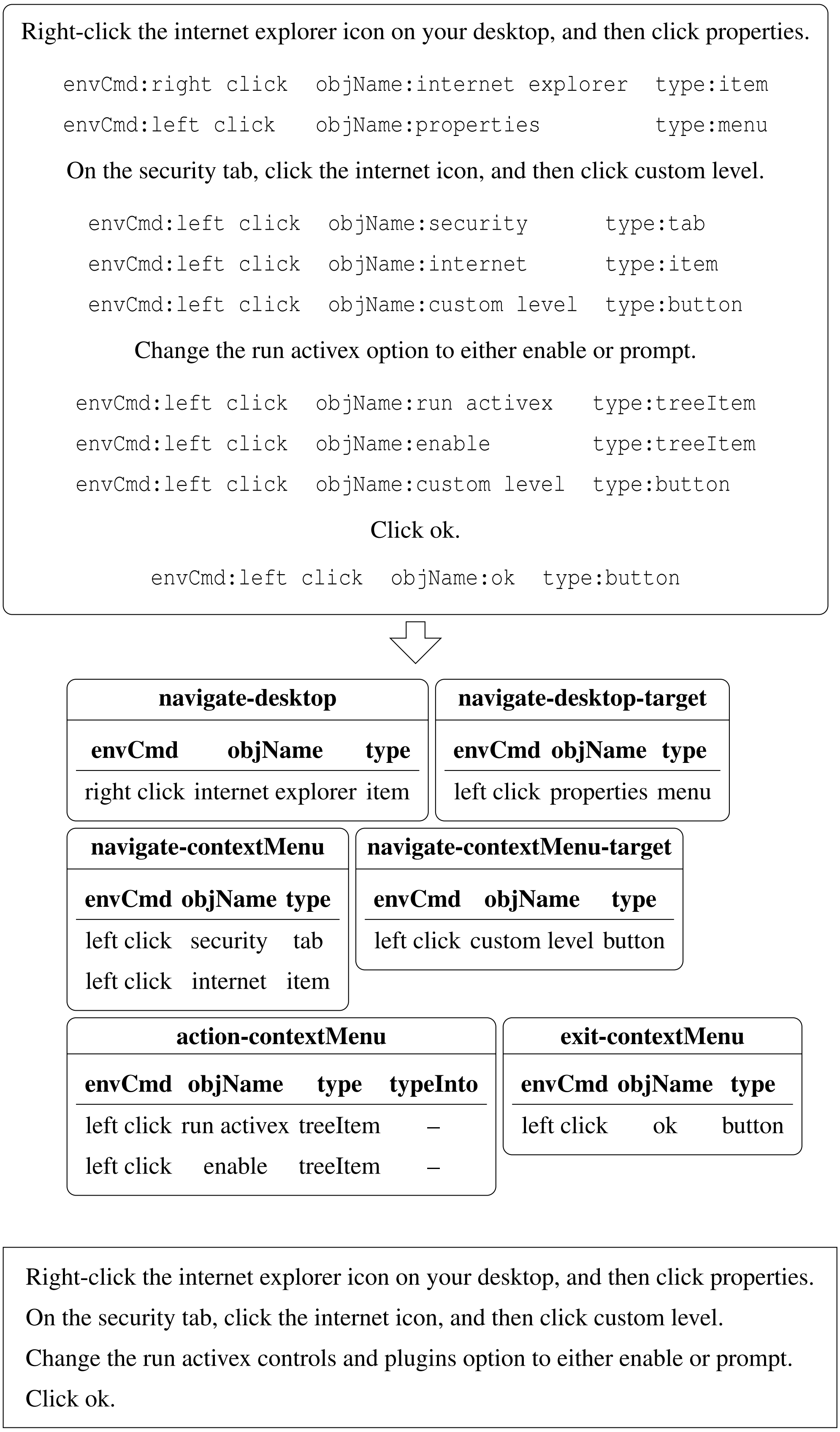
for the **Search** record of the *type* field, is `query`, and the value of the *what* field, is `flight`, since it refers to the variable x which has type flight.

3. Next we fill records with their fields and values from functions with 2 arguments, where the first argument matches with the record's type (the exception to this rule are functions with composite names; see step below). The field name comes from the name of the function and its value from the second argument, which is a constant. In the example, `from(x, denver)` is used to fill the record 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`.

4. 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 heuristically aggregate record types which embed common information (i.e., departure, or arrival) to a special field. In the example, the function `day_number_departure` becomes the value `departure` of the field *dep/ar* for the record **Day**.

5. Then we determine special **condition** record types, that correspond to the following special functions: <, >, `not` and `or`. **Condition** records have a field *type* which takes a value according to the name of the function. It also has two extra fields *arg1* and *arg2* that correspond to the function's arguments. In the example, the function `<(arrival_time(x), 1600)` is converted to a **Condition** record with values `arrival_time`, `1600` and < for the fields *arg1*, *arg2* and *type*, respectively.

In contrast to the 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 total record types mentioned on average. All the fields are of categorical type. 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 Zettlemoyer and Collins (2007), we trained on 4,962 scenarios and tested on ATIS NOV93 which contains 448 examples.



Right-click the internet explorer icon on your desktop, and then click properties.

```
envCmd:right click  objName:internet explorer  type:item
envCmd:left click   objName:properties         type:menu
```

On the security tab, click the internet icon, and then click custom level.

```
envCmd:left click  objName:security      type:tab
envCmd:left click  objName:internet      type:item
envCmd:left click  objName:custom level  type:button
```

Change the run activex option to either enable or prompt.

```
envCmd:left click  objName:run activex   type:treeItem
envCmd:left click  objName:enable        type:treeItem
envCmd:left click  objName:custom level  type:button
```

Click ok.

```
envCmd:left click  objName:ok  type:button
```

navigate-desktop

envCmd	objName	type
right click	internet explorer	item

navigate-desktop-target

envCmd	objName	type
left click	properties	menu

navigate-contextMenu

envCmd	objName	type
left click	security	tab
left click	internet	item

navigate-contextMenu-target

envCmd	objName	type
left click	custom level	button

action-contextMenu

envCmd	objName	type	typeInto
left click	run activex	treeItem	–
left click	enable	treeItem	–

exit-contextMenu

envCmd	objName	type
left click	ok	button

Right-click the internet explorer icon on your desktop, and then click properties.
On the security tab, click the internet icon, and then click custom level.
Change the run activex controls and plugins option to either enable or prompt.
Click ok.

Figure 3.6: WINHELP: Example scenario from the original log of the dataset of Branavan et al. (2009) and the resulting database schema. For a detailed description of the conversion, we refer the reader to the main text.

3.2.4 WinHelp Dataset

For the last domain (henceforth WinHelp) we created a dataset based on the troubleshooting guide corpus of Branavan et al. (2009). The authors collected articles from Microsoft's Help and Support website[5] which contain step-by-step instructions on how to perform tasks on the Windows 2000 operating system. In order to acquire some form of semantic representation, they implemented a capturing mechanism on a sandbox virtual machine running the Windows operating system, and manually performed the steps in each troubleshooting guide. This translated into navigating through a set of visible user interface (UI) objects, and object properties such as label, location, and parent window. It also included performing certain actions, i.e., left-click, right-click, double-click, and type-into (e.g., typing into a text box).

The acquired logs for each document is split into sentences and each sentence is accompanied with the set of actions required to complete the step described therein. Each action consists of an environment command, `envCmd`, the name of the object the command was operated on, `objName`, and the type of the object, `objType`. The top of Figure 3.6 gives an example of the original format.

For our purpose, we chose first to concatenate all sentences and accompanying actions into a complete document. Each action in the log is considered a separate record in our database. Then we manually annotated each action as follows:

- We assign it to a different record type, given the corresponding text and object involved. The record types are grouped into three large categories, namely **navigate-**, **action-contextMenu** and **exit-contextMenu**. The last two categories trivially correspond to actions, which involve completing the ultimate goal of the whole document, and quitting or exiting context menus, windows or programs, respectively.

- The **navigate-** category contains the main bulk of types, which describe user actions in order to access different menus and windows, before achieving the ultimate goal via an **action-contextMenu** record as described above. The possible types are **-desktop**, **-start**, **-contextMenu**, **-window**, **-program**, and **-location**.

- The **navigate-** category is further subdivided into simple navigation types (e.g., **navigate-contextMenu**) and **-target** types (e.g., **navigate-contextMenu-target**), depending on whether the action is a part of a sequence of intermediate

[5] `support.microsoft.com`

actions, or the ultimate action of the sequence which results in a change of the context in the UI environment. Usually, the former types involve clicking, or pointing the mouse over a window, or a button, whereas the latter is signified by a double-click on a button or an icon. In the example of Figure 3.6, the third and fourth records have a simple navigation type, whereas the sixth is of **-target** type. This is easily justified by the corresponding text which reads *'On the security tab, click the internet icon, and then click custom level.'*; pointing on the security tab and clicking on the internet icon are intermediate actions before the goal action, which is to double-click on the custom level button.

- Finally, the atrribute-value pairs of the original format, i.e., *envCmd*, *objName* and *objType*, become fields for our new record types; the first and third are of categorical type whereas the second is string-typed. The record type **action-contextMenu** has an extra string field called *type-into*.

The resulting dataset[6] consists of 128 scenarios. The final database has 13 record types. Each scenario has 9.2 records and each document 51.92 words with 4.3 sentences. The vocabulary is 629 words. For our experiments we performed 10-fold cross-validation on the entire dataset for training and testing. Compared to the other three datasets, WINHELP documents are longer with a larger vocabulary. More importantly, due to the nature of the domain, i.e., giving instructions, content selection is critical both in terms of what to say but also in what order.

3.3 Evaluation Methodology

In this work we address two different tasks, namely learning the alignments between the database and the text, and most importantly, generating text given the database only. In order to measure the quality of our systems we need ways to evaluate each task separately. In the following we will describe the methodology and metrics used to evaluate the alignments produced during the training of our grammar, and when generating the output text.

3.3.1 Alignment Generation Evaluation

As we will explain in more detail in Sections 4.1-4.2, an important component of the generation process is the alignment of database records to text. Although we are pri-

[6] Available from `http://homepages.inf.ed.ac.uk/ikonstas/index.php?page=resources`

marily interested in the output of our generator, the quality of the alignments unavoidably impacts the quality of the generated output, hence the overall performance of our system as well. During training, as shown at the top of Figure 3.2, we observe both the database records **d** and the text w, and try to infer the hidden correspondences or alignments between them (we describe this procedure in more detail in Section 4.4.1). Assuming gold-standard alignments between **d** and w at the record level, we can measure the quality of automatically produced ones using standard precision, recall and F_1 measure of records on the training set. Note that our model captures correspondences at a higher level of granularity, i.e., between records, fields and values, and words. However, it is prohibitively costly to obtain so detailed annotation, hence we adhere to measuring only record alignments. The procedure follows Liang et al. (2009).

For each scenario we run a Viterbi search given the input **d**, the text w and the trained grammar, and produce the best derivation tree. Then we extract from the tree a set of line-record pairs by aligning a line to a record r_i, if the span of the phrase segment corresponding to r_i, overlaps the line. In ROBOCUP and ATIS we consider the whole comment or utterance as a line, respectively. For WINHELP we split lines into sentences as they were segmented in the original dataset. Finally, for WEATHERGOV the text is split into lines at punctuation (if there are any), rather than at phrase segments, since they are easier to obtain automatically in a consistent way. We provide the definition of Precision, Recall and F_1 for reference:

$$Precision = \frac{Correctly\ Aligned\ Records}{Correctly\ Aligned\ Records + Extra\ Aligned\ Records} \tag{3.1}$$

$$Recall = \frac{Correctly\ Aligned\ Records}{Correctly\ Aligned\ Records + Missing\ Records} \tag{3.2}$$

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{3.3}$$

Figure 3.7 gives an example of gold-standard and generated record alignments; in red we highlight the records mismatches. In order to calculate precision we divide the number of correctly aligned records (4) by the sum of correctly aligned plus extra aligned records (5), which equals to 0.8. Recall and F_1 are computed accordingly $\left(\frac{4}{7} = 0.57\right)$, and $\left(2 \cdot \frac{0.8 \cdot 0.57}{0.8+0.57} = 0.67\right)$, respectively.

3.3.2 Generation Evaluation

Deciding whether a text is of good quality is a difficult and often too subjective task to be handled automatically. However, there are some widely used automatic metrics

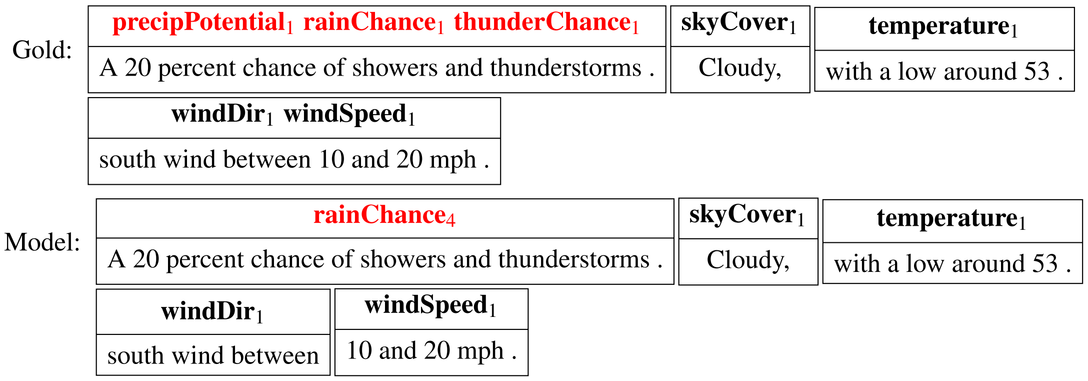


Figure 3.7: Gold-standard and generated record alignments. In red we highlight the mismatched records.

such as BLEU score (Papineni et al., 2002) which assess the quality of a generated text compared to (usually a human-authored) reference, and have been shown to correlate well with human judges. Many data-driven approaches advocate its use, such as Belz (2008); Belz and Reiter (2006); Angeli et al. (2010), inter alia. The use of automatic metrics can also be helpful while developing an algorithm, or during parameter tuning. We therefore evaluate the output of our system in two ways: using widely accepted automatic measures of surface level output, and by eliciting human judgement studies.

The first metric we use is the BLEU score (Papineni et al., 2002) with the human-written text as reference. The main component of BLEU is n-gram precision, i.e., the proportion of the matched n-grams in the gold-standard text out of the total number of n-grams in the evaluated generated text. Precision is calculated separately for each n-gram order, and the precisions are combined via a geometric averaging and a brevity penalty that penalises shorter or longer generated output compared to the original. We take into account up to 4-grams (hence the metric is referred to as BLEU-4), which is standard practice in the Machine Translation community. More formally:

$$BLEU = BP \cdot exp\left(\sum_{n=1}^{N} w_n log p_n\right) \tag{3.4}$$

where BP is a brevity penalty function following an exponential distribution, w_n are the weights for each n-gram set to follow a geometric distribution, and p_n is the modified n-gram precision. We implemented our own version of the BLEU score metric.

The second metric we use is the METEOR score (Banerjee and Lavie, 2005; Denkowski and Lavie, 2011), which has been shown to correlate better with human judgements at the sentence level. In contrast with BLEU, METEOR computes unigram matches between the gold and the generated text based on either the surface form, the stemmed form or even the meaning, i.e., if they are synonyms according to the WordNet (Miller, 1995) database. METEOR is the combination of unigram precision, unigram recall (not present in BLEU), and a metric of fragmentation, which directly captures the degree of matching words that are well-ordered (this replaces the fixed brevity penalty of BLEU). We omit the formal description of the metric from here and refer the interested reader to Banerjee and Lavie (2005) for more details. We used the existing implementation of METEOR obtained from `www.cs.cmu.edu/~alavie/METEOR/`.

Finally, we evaluate the generated text via judgement elicitation studies, in which participants are 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?). Subjects in our experiments use a five point rating scale where a high number indicates better performance. We conducted our studies over the Internet using Amazon Mechanical Turk (AMT)[7], and all participants were self reported native English speakers.

AMT is an online labour market where workers are paid small amounts of money to complete small tasks. There are two types of users: ones who submit Human Intelligence Tasks (or HITs) for annotation and others who actually annotate the submitted task; both are required to have an Amazon account, but appear as anonymous in the platform. The Requesters define the number of unique annotations per HIT they permit, the maximum time limit they allow for each Worker to annotate the task, and the total payment per task. AMT also allows a Requester to restrict which Workers are allowed to annotate a task by requiring that all Workers have a particular set of qualifications (e.g., a minimum percentage of previously accepted submissions). Finally, when a HIT is completed, the Requester is given the option to approve the work of individual Workers.

3.4 Summary

In this chapter we defined our generation task, described the input to our model and highlighted the important parts of our system architecture. We then presented an overview of the four domains we experimented on, and discussed each database schema we used. Finally, we summarised the evaluation methodology we adopted to assess the performance of our systems, and the specific metrics we used.

[7] `www.mturk.com`

Chapter 4

Joint Model of Generation

In this chapter we propose a model for performing jointly content selection, rudimentary sentence planning (only lexicalisation) and surface realisation. We begin by presenting in detail an existing model for content selection from database input (Liang et al., 2009), which we then extend and recast as a PCFG grammar. Next, we demonstrate several ways to efficiently parse with the grammar in order to generate fluent text, and conclude with an extensive evaluation on the four domains presented in the previous chapter.

4.1 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. 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.

By concatenating the sequences of word choices for each field, we retrieve the observed text $\mathbf{w}$. Note that the segmentation of $\mathbf{w}$ into sequences of c_{ik} words is latent.

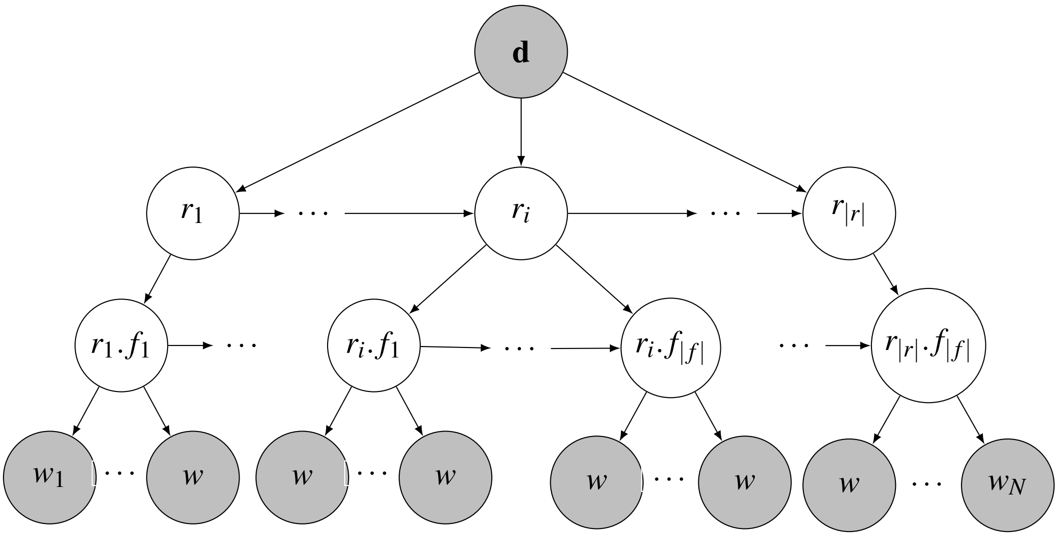


Figure 4.1: Graphical model representation of the generative alignment model of Liang et al. (2009). Shaded nodes represent observed variables (i.e., the database $\mathbf{d}$ and the collocated text w), unshaded nodes indicate latent variables. Arrows indicate conditional dependencies between variables. Starting from the database $\mathbf{d}$, 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 $w_1 \dots w_c$.

The process described above is implemented as a hierarchy of Markov chains which correspond to records, fields, and values of the input database. As can be seen in Figure 4.1 the choice of records, is 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:

$$p(\mathbf{r}\,|\,\mathbf{d}) = \prod_{i}^{|\mathbf{r}|} p(r_i.t\,|\,r_{i-1}.t)\frac{1}{|\mathbf{s}(r_i.t)|} \tag{4.1}$$

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 modelled analogously as a Markov chain of fields for a given record choice r_i of type t:

$$p(\mathbf{f}\,|\,r_i.t) = \prod_{k}^{|r_i.\mathbf{f}|} p(r_i.f_k\,|\,r_i.f_{k-1}) \tag{4.2}$$

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, they do not need a more powerful representation at the surface level:

$$p(\mathbf{w}\,|r_i, r_i.f_k, r_i.f_k.t, c_{ik}) = \prod_{j}^{|\mathbf{w}|} p(w_j\,|\,r_i.t, r_i.f_k.v) \tag{4.3}$$

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 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 of 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 modelled as a geometric distribution, the parameters of which are trained given the value $r_i.f_k.v$.

Records:	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*	*temps between*	*10*	*20 degrees .*	*south*	*wind*	*around*	*20 mph .*

Figure 4.2: Example of alignment output for the model of Liang et al. (2009) on the weather domain.

An example of the model's output for the weather domain is shown in Figure 4.2. 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 4.3). 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 and 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 spanned by the chosen records and fields. The generation process is then modelled as a series of local decisions, arranged hierarchically and each trained discriminatively. They first choose which records to talk about, then a subset of fields for each record, and finally a suitable template to render the chosen content.

We do not treat 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.

4.2 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 4.1. 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 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. (2009), namely it also describes a regular language.

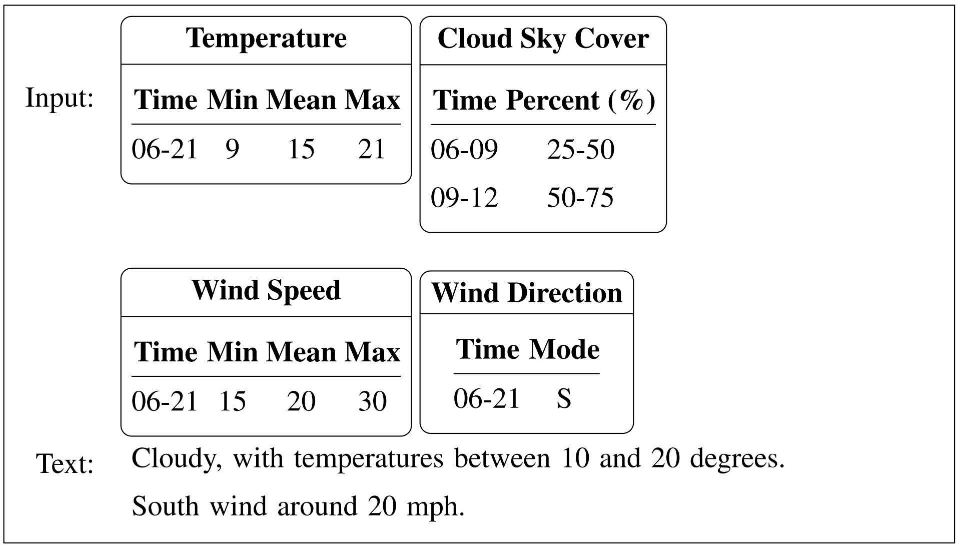
Our grammar G_{GEN} is defined in Table 4.1 (rules (1)–(10)) and contains two types of rules. G_{CS} rules perform content selection, whereas G_{SURF} rules perform surface realization. We do not explicitly model the process of sentence planning, as described in Section 2.1, nor do we deal with the more sophisticated aspects of referring expression generation and aggregation. However, we indirectly model the lexicalisation of database values of fields, simultaneously with surface realisation. All 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. We estimate rule weights in an unsupervised fashion using EM (see Section 4.4.1). Non-terminal symbols are in capitals and denote intermediate states; the terminal symbol α corresponds to a single word from the set of 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 4.4 shows two derivation trees licensed by our grammar for the sentence "*Cloudy, with temperatures between 10 and 20 degrees.*" (see the example in Figure 4.3).

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 $\mathrm{R}(start)$). 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 4.4a, the top branching rule $\mathrm{R}(start) \rightarrow \mathrm{FS}(sc_2, start)\mathrm{R}(sc_2.t)$ (sc

G_{CS}	1. $\mathrm{S} \rightarrow \mathrm{R}(start)$	$[Pr = 1]$
	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. $\mathrm{W}(r, r.f) \rightarrow \mathrm{gen}(f.v)$	$[P(\mathrm{gen}(f.v).mode \mid r, r.f, f.t = int) \cdot$ $P(f.v \mid gen(f.v).mode)]$
	10. $\mathrm{W}(r, r.f) \rightarrow gen\_str(f.v, i)$	$[Pr = 1]$

Table 4.1: Grammar rules for G_{GEN} and their weights shown in square brackets.

stands for **Cloudy Sky Cover**) can be interpreted as follows. Given we are at the beginning of the document, hence the record $\mathrm{R}(start)$, we will talk about the part of the forecast that refers to **Cloud Sky Cover**, i.e., emit the set of fields spanned by the non-terminal $\mathrm{FS}(sc_2, start)$. The field *start* in FS acts as a special boundary between consecutive records. Note that in the input database of example 4.3, there are five records of type **Cloud Sky Cover**. Given that the value of the *Percent (%)* field of the second record is 50-75, it is more likely to lexicalise 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 $\mathrm{R}(sc_2.t) \rightarrow \mathrm{FS}(t_1, start)\mathrm{R}(t_1.t)$ (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 $\mathrm{FS}(t_1, start)$ (see the second sub-tree in Figure 4.4a). The weight of this rule is the bigram probability of two records conditioned on their record type, multiplied with the normalization factor $\frac{1}{\lvert \mathbf{s}(r_i.t) \rvert}$, 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



Input:

Temperature

Time	**Min**	**Mean**	**Max**
06-21	9	15	21

Cloud Sky Cover

Time	**Percent (%)**
06-09	25-50
09-12	50-75

Wind Speed

Time	**Min**	**Mean**	**Max**
06-21	15	20	30

Wind Direction

Time	**Mode**
06-21	S

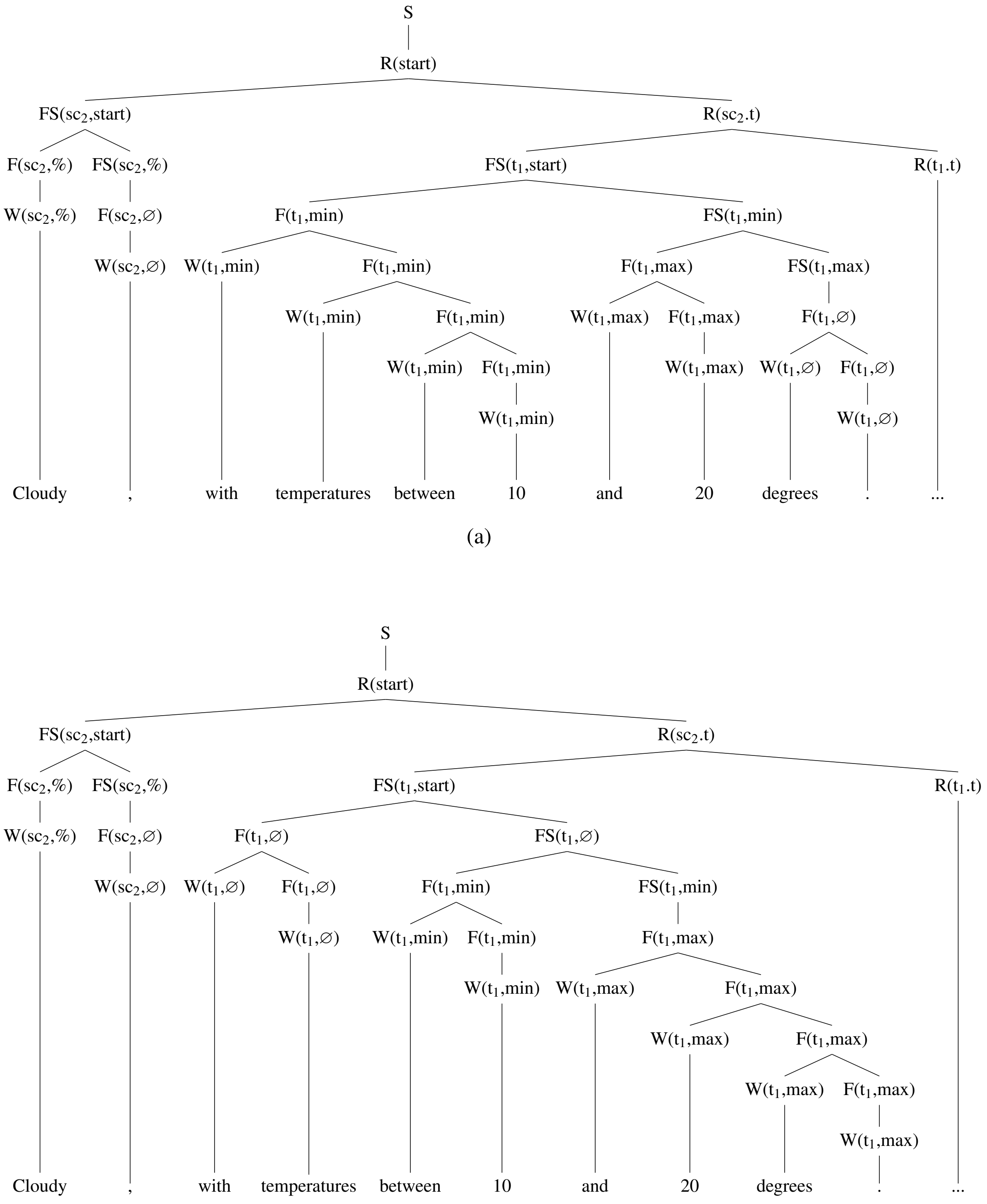
Text: Cloudy, with temperatures between 10 and 20 degrees.
South wind around 20 mph.

Figure 4.3: Example scenario on the WEATHERGOV domain.

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 $\mathrm{F}(r, r.f)$ refers to field f of record r. For example, in the tree of Figure 4.4a, the rule $\mathrm{FS}(t_1, min) \rightarrow \mathrm{F}(t_1, max)\ \mathrm{FS}(t_1, max)$ 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)-(10) are responsible for lexicalisation and 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 4.4a 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



(a)

(b)

Figure 4.4: Two derivation trees using the grammar in Table 4.1 for the sentence "*Cloudy, with temperatures between 10 and 20 degrees.*" of the scenario in Figure 4.3. We use *sc* as a shorthand for the record type **Cloudy Sky Cover**, and *t* for **Temperature**. Subscripts refer to record tokens (e.g., sc_2 is the second Cloudy Sky Cover record, t_1 is the first Temperature record, and so on).

type is integer. Function 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. Finally, rule (10) adds a simple verbatim lexicalisation for string values. We define gen_str as a function that takes the value of a string-typed field $f.v$, and the position i in the string, and generates the corresponding word at that position:

$$\text{gen\_str}(f.v, i) : V \to V,\ f.v \in V$$

where V is the set of words for the fields of type string. Taking an example from the WINHELP domain, gen_str(*users and passwords*, 3) = *passwords*. The weight of this rule is set to 1.

4.3 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 multinomial distributions (see the rule weights of G_{GEN} in Table 4.1) define a hidden correspondence h between records, fields and their values. Given an input scenario from a database **d** we can simply generate its corresponding text using the grammar in Table 4.1. In analogy to parsing, this 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, the string is not observed; instead, we must thus find the best text $\hat{g}$, by maximizing both over h and g[1], where $g = g_1 \dots g_N$ is a sequence of words licensed by G_{CS} and G_{SURF}. More formally:

$$g = f\Big(\arg\max_{g,h} P\big((\hat{g}, \hat{h})\big)\Big) \tag{4.4}$$

[1] We use w to denote the gold-standard text and g to refer to the string of words produced by our generator.

where f is a function that takes as input a derivation tree (g,h) and returns 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 4.3.2. An additional complication is that since we do not a priori know the length of our output text, we must somehow approximate or guess it. We defer discussion on how we achieve this to Section 4.4.3.

In the following, we describe our decoder as a deductive proof system (Shieber et al., 1995). We first present a basic adaptation of the CYK algorithm to our task (Section 4.3.1) and then extend it by integrating external linguistic knowledge in an attempt to improve the quality of the output. The basic decoder 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 provides a level of abstract generalization for a number of algorithms. Examples include the recognition 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). Also note that our methodology is similar in spirit to the chart realisation frameworks presented in Section 2.2.1. We also transduce our database input schema to strings using a chart parser. However, both our grammar and parsing strategy inherently dictates the selection of particular database records, fields and values before realising them, thus departing from the idea of strict surface realisation from a given input logical form. Finally, existing chart realisers rely on some form of initial lexicon that contains bilexical entries, that couple semantics with surface forms (it is also common for surface forms to also contain extra features such as tense, number, gender, and so on). In our framework, we instead implicitly jointly infer our own lexicon as part of our grammar in the form of rules in G_{SURF} .

4.3.1 Basic Decoder

A parser 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

Items: $[A, i, j]$

$R(A \rightarrow BC)$

Axioms: $[A, i, i+1] : s \qquad A \rightarrow w_{i+1}$

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

Goal: $[S, 0, N]$

Figure 4.5: The CYK algorithm for any CFG in Chomsky normal form and input string $w = w_1 \ldots w_N$.

set of axioms, and progressively applies the inference rules, in order to prove more items until it reaches one of the designated goals.

For example the CYK algorithm for context-free grammars in Chomsky normal form, 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 (Figure 4.5). Following Goodman (1999), items take two forms: $[A, i, j]$ indicates a generated span from i to j, rooted at non-terminal A; $R(A \rightarrow B\ C)$ corresponds to any of the production rules of the grammar with two non-terminal symbols on the right hand side. Axioms correspond to each individual word generated by the lexical grammar rules $A \rightarrow \alpha$, where α is a terminal symbol. The goal of the proof system is the special item $[S, 0, N]$, where S is the root node of the grammar and N the length of the generated text.

Our basic decoder is similarly specified in Figure 4.6. Items in our system also take two forms, namely $[A, i, j]$ as above, and $R(A \rightarrow B)$ or $R(A \rightarrow B\ C)$ corresponding 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 (8) (10) in G_{SURF}. Our inference rules follow two forms, one for grammar production rules with one non-terminal on the right hand side, and another for rules with two non-terminals. For example, inference rule (1) in Figure 4.6 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 in a new generated span $[A, i, j]$ with weight $s \cdot s_1$, rooted at A. Finally, our system has a goal similar to CYK, $[S, 0, N]$, where N is 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 into Chomsky normal form (CNF) and implemented the original CYK algorithm. We chose to directly implement inference rules (1) and (2) instead, 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.

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

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

Inference rules:

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

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

Goal: $[S,0,N]$

Figure 4.6: The basic decoder deductive system. The production rules $A \to B$ and $A \to B\ C$ are any of the set G_{CS}; features on grammar non-terminals are omitted here for the sake of clarity.

Now that we have defined the parsing strategy, we need a way to find the most likely derivation; the pseudocode of Figure 4.7 gives the Viterbi search procedure 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. The procedure begins by filling in the cells of the *chart* with unary span rooted at W, with the weights of the lexical rules $r \in G_{SURF}$. Equivalently, the back-pointers array takes the corresponding generated word. 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.

4.3.2 *k*-best Decoding

The basic decoder described so far will produce the best derivation tree of the input $\mathbf{d}$ 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 Viterbi search process in Figure 4.7 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 of Huang and Chiang (2005). We can then use a language model such as higher order n-grams, or head dependency-style rules to rescore the generated strings directly (see also Charniak and Johnson (2005) and Liang et al. (2006) for application of a similar idea to parsing and machine translation, 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

```
function DECODE($G_{GEN}$,$\mathbf{d}$,$N$)
    for $i \leftarrow 0 \ldots N$ do
        for all $r \in \mathbf{d} : W \rightarrow g_{i+1} \in G_{SURF}$ do
            $chart[W,i,i+1] \leftarrow [W,i,i+1] : s$
            $bp[W,i,i+1] \leftarrow g_{i+1}$
        end for
    end for
    for $l \leftarrow 2 \ldots N$ do
        for all $i,k,j$ so that $j - i = l$ and $i < k < j$ do
            for all items $[B,i,j]$ or $[B,i,k]$, $[C,k,j]$ inferrable from $chart$ and rules $r \in G_{CS}$ do
                if r is of the form $A \rightarrow B$ then
                    $chart[A,i,j] \leftarrow max([B,i,j] : s_1 \times P(r))$
                    $bp[A,i,j] \leftarrow argmax([B,i,j] : s_1 \times P(r))$
                end if
                if r is of the form $A \rightarrow B\ C$ then
                    $chart[A,i,j] \leftarrow max(chart[B,i,k] \times chart[C,k,j] \times P(r))$
                    $bp[A,i,j] \leftarrow argmax([B,i,k] : s_1 \times [C,k,j] : s_2 \times P(r))$
                end if
            end for
        end for
    end for
    return $chart[S,0,N]$, $bp[S,0,N]$
end function
```

Figure 4.7: Viterbi Search procedure for the basic decoder.

of external probabilistic models, provided that they express a regular language. The most probable generation $\hat{g}$ is then calculated as:

$$\hat{g} = f\Big(\arg\max_{g,h} p(g) \cdot p(g, h \,|\, \mathbf{d})\Big) \tag{4.5}$$

where $p(g,h\,|\,\mathbf{d})$ is the decoding likelihood for a sequence of words $g = g_1 \ldots g_N$ 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 4.4.1 for details on how we estimate $p(g,h\,|\,\mathbf{d})$ and $p(g)$). In theory the function f above should optimise 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 4.6 by intersecting it (linearly) with an ensemble of external probabilistic models.

4.3.2.1 Intersection with External Models

A n-gram language model is an $n-1$-th order Markov chain, the states of which are words. Given a sentence $w = w_1 \ldots w_N$ of length N, with $w_i \ldots w_{i-1} \in \mathcal{V}$, $w_N = \langle /s \rangle$ a special stop symbol, and $\mathcal{V}$ the vocabulary of the language, the probability in particular of a 2nd order Markov chain language model, or 3-gram language model is:

$$P(w_1 \ldots w_N) = \prod_{i=1}^{N} q(w_i \,|\, w_{i-1}, w_{i-2})$$

where $w_0 = w_{-1} = \langle s \rangle$, a special start symbol for the sentence. The estimates q of the trigrams are obtained from the training corpus, usually incorporating some kind of smoothing such as Good-Turing (Good, 1953) or Kneser-Ney (Kneser and Ney, 1995) and backing-off techniques to lower order models for unseen trigrams (Katz, 1987).

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 4.8 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 4.8b 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 time determines the behavior of the pair. In Figure 4.8b, *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., $D(\mathit{10}) = \{\mathit{and},\ \mathit{20}\}$.

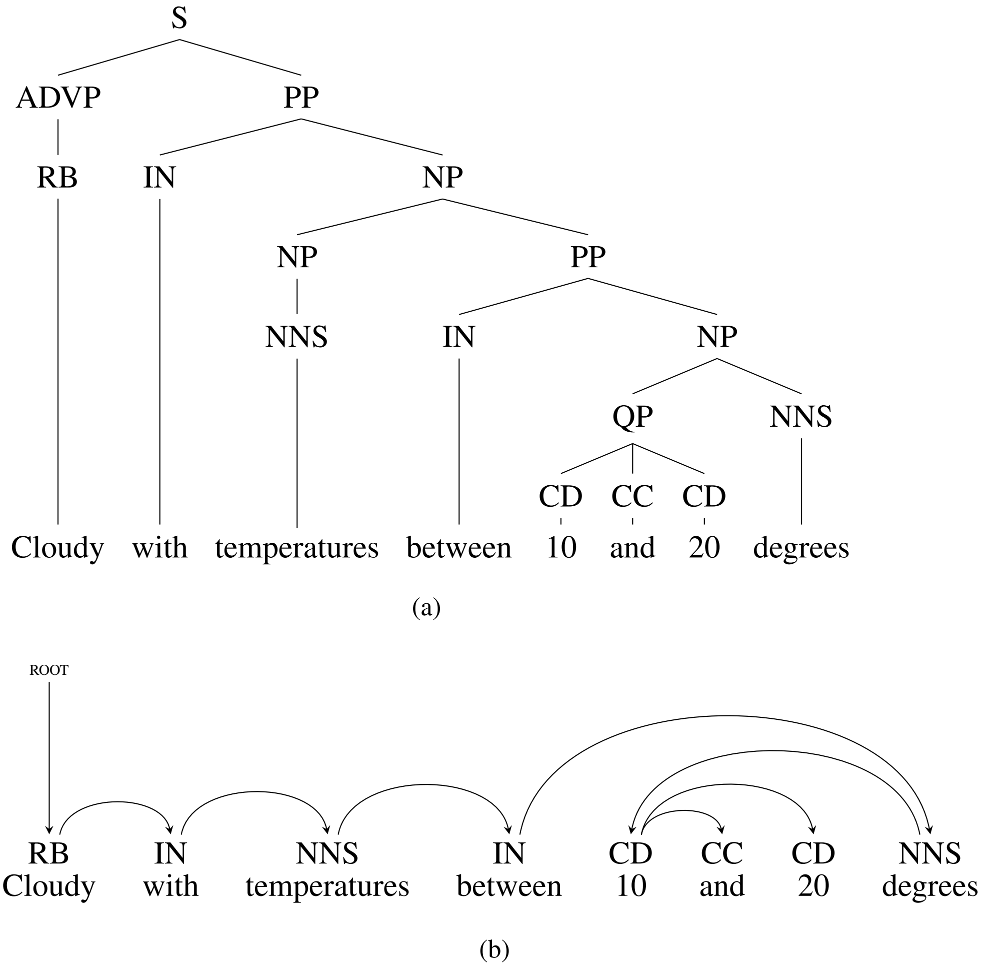


Figure 4.8: Phrase structure tree and dependency graph for the same sentence.

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 and 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 is weakly equivalent with our grammar, i.e., it generates the same surface string regardless of the possibly different dependency structures it may induce. Furthermore, its expressive power should not exceed that of a regular language, which is the case for the DMV, as our aim is to intersect it with a CFG.[2]

DMV (Klein and Manning, 2004) is defined as a head-outward dependency model over word classes, in our case part-of-speech (POS) tags derived from the Penn Treebank project (Marcus et al., 1993), which includes a model of valence. In other words they formulate a non-recursive PCFG that imposes a search strategy from the head of a word (or equivalently the class of the word) to its dependent arguments, taking into consideration the distance of the arguments to the head (the farther away a word is from the head, the less probable it is to be attached as an argument). The generative process they describe begins at the ROOT word. Then each head generates a series of non-STOP arguments to one side (i.e., left or right), then a STOP argument to that side, then a sequence of non-STOP arguments to the other side and finally a second STOP argument.

For example, in the dependency structure in Figure 4.8b, under this process, we first generate a single child of ROOT, here *Cloudy*[3]. Then we recurse to the subtree under *Cloudy*. This subtree first generates the right argument *with*. The recursion continues to the subtree under *with*, and likewise under *temperature*, *between* and *degrees*. After the word *degrees* it generates a right STOP, since there is no word on its right that could get attached as a dependent-argument, and starts generating on its left with the word *10*. The process continues in the same fashion until the word *20*; there it generates a right STOP, then a left STOP and since there is no other word in either direction left to attach, the process ends.

Finally, note that although we work with two external information sources (i.e.,

[2] Intersecting two CFGs is undecidable, or PSPACE-complete if one CFG is finite (Nederhof and Satta, 2004).

[3] In this example we use a lexicalised version of the process, for demonstration purposes only. The exact same procedure applies, by replacing words with the POS tags.

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.

4.3.2.2 Notation

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 4.9) on the output of $q(a)$ allows us to correctly compute the rescoring weight of each model for the whole document incrementally. More formally:

$$p(a) = \sum_i^M \beta_i p_{m_i}(a) \qquad \text{s.t.} \sum_i^M \beta_i = 1 \tag{4.6}$$

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

$$p_{m_1}(a_1 \dots a_l) = \prod_{\substack{n \le i \le l \\ \star \notin \{a_{i-n+1}, \dots, a_i\}}} P_{LM}(a_i | a_{i-n+1} \dots a_{i-1}) \tag{4.7}$$

$$p_{m_2}(a_1 \dots a_l) = P_{DEP}\big(D(a_h)\big), \text{ where } a_h \in \{a_1, \dots, a_l\} \tag{4.8}$$

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 dependency model on the dependency structure D headed by word a_h. For a dependency structure D, each word a_h has dependents $deps_D(a_h, left)$

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

Table 4.2: 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.

that attach on its left and dependents $deps_D(a_h, right)$ that attach on its right. Equation (4.9) recursively defines the probability of the dependency $D(a_h)$ rooted at a_h Klein and Manning (2004):

$$\begin{aligned} P_{DEP}\big(D(a_h)\big) = \prod_{dir \in [left, right]} \Big[\prod_{deps_D(a_h, dir)} & P_{\text{STOP}}(\neg\text{STOP}|a_h, dir, adj) \\ & P_{\text{CHOOSE}}(a_a|a_h, dir) P_{DEP}\big(D(a_a)\big)\Big] \\ & P_{\text{STOP}}(\text{STOP}|a_h, dir, adj) \end{aligned} \tag{4.9}$$

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 4.6).

$$q(a) = \langle q_{m_1}(a), \dots, q_{m_M}(a) \rangle \tag{4.10}$$

(4.11)

$$q_{m_1}(a_1 \dots a_l) = \begin{cases} a_1 \dots a_{n-1} \star a_{l-n+2} \dots a_l & \text{if } l \geq n \\ a_1 \dots a_l & \text{otherwise} \end{cases} \tag{4.12}$$

(4.13)

$$\underset{1 \leq k \leq l}{\mathrm{q_{m_2}}} (a_1 \dots a_k a_{k+1} \dots a_l) = \begin{cases} a_l & \text{if } l = 1 \\ q_{m_2}(a_1 \dots a_k) & \text{if } p_{m_2}(a_1 \dots a_k) \geq \\ & \quad p_{m_2}(a_{k+1} \dots a_l) \\ q_{m_2}(a_{k+1} \dots a_l) & \text{otherwise} \end{cases} \tag{4.14}$$

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 4.2 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 sub-strings $(a_1 \dots a_k)$ 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.

Note that equation (4.14) evaluates whether every word should attach to the left or right of every other head word, and therefore essentially collapses to:

$$P_{m_{dep}} = P_{DEP}\big(D(a_h)\big) = P_{\text{STOP}}(\neg\text{STOP}|a_h, dir, adj) P_{\text{CHOOSE}}(a_a|a_h, dir) \\ P_{\text{STOP}}(\text{STOP}|a_h, dir, adj) \tag{4.15}$$

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 $a_a = a_{k+1}$.

Items: $[A, i, j; q(g_i^j)]$

$R(A \rightarrow B)$

$R(A \rightarrow BC)$

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

Inference rules:

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

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

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

Figure 4.9: Extended decoder using the rescoring function $p(g)$. Productions $A \rightarrow B$ and $A \rightarrow B\ C$ can be any of the G_{CS} rules in Figure 4.1; features on grammar non-terminals are omitted for the sake of clarity.

4.3.2.3 Extended Decoder

We are now ready to extend the basic decoder in Figure 4.6, 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 4.9. Items $[A,i,j]$ become now $[A,i,j;q(g_i^j)]$; they represent derivations spanning 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 4.10 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 4.7, save the exponentially more items that need to be deduced. Recall that the *chart* of the Viterbi search for the basic decoder in Figure 4.7 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, 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 4.9 combines two items that contain at most $2(n-1)$ words, hence the exponent $4(n-1)$. 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.

4.3.2.4 Approximate Search

Consider the task of deriving a k-best list of items $\mathbf{L}([A,i,j;q(g_i^j)])$ for the deduced item $[A,i,j;q(g_i^j)]$ of rule (2) in the extended decoder of Figure 4.9. 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 4.11. 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

$$\frac{\begin{array}{c} R\,(\mathrm{R}(skyCover_1.t) \to \mathrm{FS}(temp_1,start)\ \mathrm{R}(temp_1.t)) : s \\ [\mathrm{FS}(temp_1,start),1,2;\langle \text{with, IN}\rangle] : s_1\ \ [\mathrm{R}(temp_1.t),2,8;\langle \text{a low} \star 15 \text{ degrees, JJ}\rangle] : s_2 \end{array}}{[\mathrm{R}(skyCover_1.t),1,8;\langle \text{with a} \star 15 \text{ degrees, JJ}\rangle] : s \cdot s_1 \cdot s_2 \cdot p(\langle \text{with a} \star 15 \text{ degrees, JJ}\rangle)}$$

$$\frac{\begin{array}{c} R\,(\mathrm{FS}(windSpeed_1,min) \to \mathrm{F}(windSpeed_1,max)\ \mathrm{FS}(windSpeed_1,max)) : s \\ [\mathrm{F}(windSpeed_1,max),3,4;\langle \text{high, JJ}\rangle] : s_1\ \ [\mathrm{FS}(windSpeed_1,max),4,5;\langle \text{15, CD}\rangle] : s_2 \end{array}}{[\mathrm{FS}(windSpeed_1,min),3,5;\langle \text{high 15, JJ}\rangle] : s \cdot s_1 \cdot s_2}$$

$$\frac{R\,(\mathrm{F}(windDir_1,mode) \to \mathrm{W}(windDir_1,mode)) : s\ \ [\mathrm{W}(windDir_1,mode),3,4;\langle \text{southeast, JJ}\rangle] : s_1}{[\mathrm{F}(windDir_1,mode),3,4;\langle \text{southeast, JJ}\rangle] : s \cdot s_1}$$

Figure 4.10: Inference rules in the extended decoder for productions (2), (4), and (7) from Table 4.1 (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*.

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 three dimensions, one for all the rules $A \to B\ C$ with the same left hand-side non-terminal A, and two for the corresponding items rooted on B and C[4]; this calls for the calculation of k^3 combinations. A better approach is to apply *cube pruning* (Chiang, 2007; Huang and 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.

Consider Figure 4.12 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 4.12b–4.12d illustrate the enumeration of

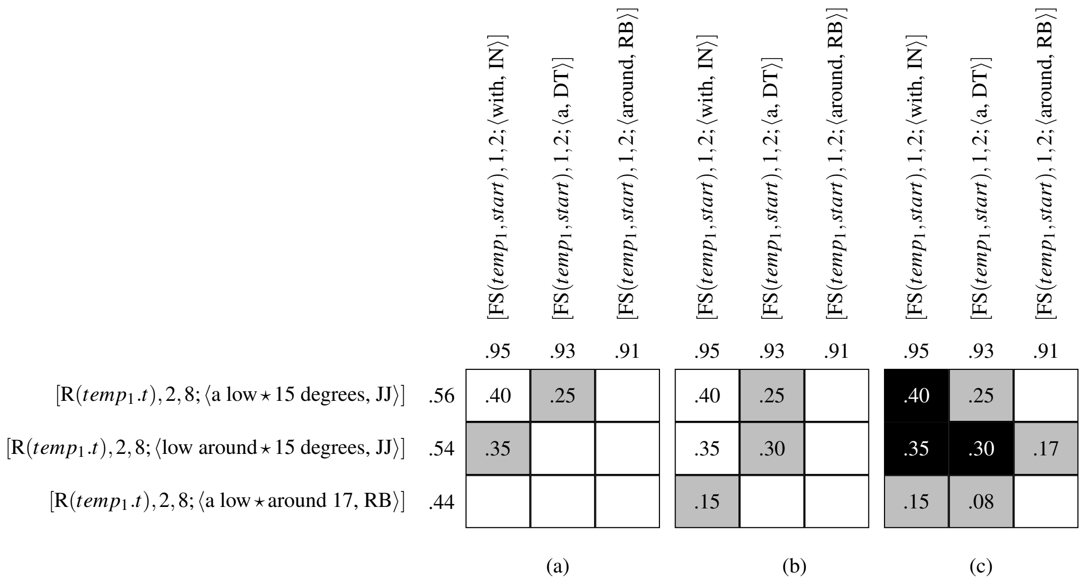
[4] The deduced item $[R(skyCover_1.t); q(g_1^8)]$ of Figure 4.11 can also be inferred by the rule $R(R(skyCover_1.t) \to R(windSpeed_1.t)\ FS(windSpeed_1,start))$ (and its corresponding antecedent items) or the rule $R(R(skyCover_1.t) \to R(rainChance_1.t)\ FS(rainChance_1,start))$, and so on. We illustrate only a slice of the cube, depicting the enumeration of k-best lists for a fixed grammar rule, for the sake of clarity.

		[FS($temp_1, start$), 1, 2; ⟨with, IN⟩]	[FS($temp_1, start$), 1, 2; ⟨a, DT⟩]	[FS($temp_1, start$), 1, 2; ⟨around, RB⟩]
		.95	.93	.91
[R($temp_1.t$), 2, 8; ⟨a low ⋆ 15 degrees, JJ⟩]	.56	.40	.25	.20
[R($temp_1.t$), 2, 8; ⟨low around ⋆ 15 degrees, JJ⟩]	.54	.35	.30	.17
[R($temp_1.t$), 2, 8; ⟨a low ⋆ around 17, RB⟩]	.44	.15	.08	.10

$$\Downarrow$$

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

Figure 4.11: Computing an exhaustive list for the deduced item $[R(skyCover_1.t); q(g_1^8)]$ via application of inference rule (2) of the extended decoder in Figure 4.10. The antecedent items are the rule $R\,(R(skyCover_1.t) \rightarrow R(temp_1.t)\ FS(temp_1, start))$ and the items $[R(temp_1.t), 2, 8; q(g_2^8)]$, $FS(temp_1, start), 1, 2; q(g_1^2)]$. The figure shows all the different item combinations 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.



(a)

		$[\mathrm{FS}(temp_1,start),1,2;\langle$with, IN$\rangle]$ .95	$[\mathrm{FS}(temp_1,start),1,2;\langle$a, DT$\rangle]$ .93	$[\mathrm{FS}(temp_1,start),1,2;\langle$around, RB$\rangle]$ .91
$[\mathrm{R}(temp_1.t),2,8;\langle$a low$\star$15 degrees, JJ$\rangle]$	.56	.40	.25	
$[\mathrm{R}(temp_1.t),2,8;\langle$low around$\star$15 degrees, JJ$\rangle]$	.54	.35		
$[\mathrm{R}(temp_1.t),2,8;\langle$a low$\star$around 17, RB$\rangle]$	.44			

(b)

		$[\mathrm{FS}(temp_1,start),1,2;\langle$with, IN$\rangle]$ .95	$[\mathrm{FS}(temp_1,start),1,2;\langle$a, DT$\rangle]$ .93	$[\mathrm{FS}(temp_1,start),1,2;\langle$around, RB$\rangle]$ .91
$[\mathrm{R}(temp_1.t),2,8;\langle$a low$\star$15 degrees, JJ$\rangle]$	.56	.40	.25	
$[\mathrm{R}(temp_1.t),2,8;\langle$low around$\star$15 degrees, JJ$\rangle]$	.54	.35	.30	
$[\mathrm{R}(temp_1.t),2,8;\langle$a low$\star$around 17, RB$\rangle]$	.44	.15		

(c)

		$[\mathrm{FS}(temp_1,start),1,2;\langle$with, IN$\rangle]$ .95	$[\mathrm{FS}(temp_1,start),1,2;\langle$a, DT$\rangle]$ .93	$[\mathrm{FS}(temp_1,start),1,2;\langle$around, RB$\rangle]$ .91
$[\mathrm{R}(temp_1.t),2,8;\langle$a low$\star$15 degrees, JJ$\rangle]$	.56	.40	.25	
$[\mathrm{R}(temp_1.t),2,8;\langle$low around$\star$15 degrees, JJ$\rangle]$	.54	.35	.30	.17
$[\mathrm{R}(temp_1.t),2,8;\langle$a low$\star$around 17, RB$\rangle]$	.44	.15	.08	

Figure 4.12: Computing item combinations for items $u_1 = [R(temp_1.t), 2, 8; q(g_2^8)]$, $u_2 = [FS(temp_1, start), 1, 2; q(g_1^2)]$ 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.

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 $u_1 = [R(temp_1.t), 2, 8; q(g_2^8)]$ and $u_2 = [FS(temp_1, start), 1, 2; q(g_1^2)]$ 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[5]:

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

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 4.12a-c, starting from $(L_1(u_1), L_1(u_2))$ and

[5] Note that the head of the sub-generation fragment has shifted to the head of L_2.

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 4.12b it is the latter). We thus select it and move on to compute its neighboring combinations, and so on.[6]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 the computed item combinations in a temporary buffer, and then resort it after we have enumerated a total of k combinations.

4.3.3 Hypergraph Representation

We represent our grammar and each input scenario as a weighted hypergraph (Gallo et al., 1993). 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 et al., 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 et al., 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[7] and the fact that it allows for 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. In fact in Chapter 5 we extend our grammar with a set of context-free rules, hence finite state implementations are not suitable any more. In the following we first give the definition of hypergraphs, and then describe an automatic process to convert the basic and the extended decoder, presented in the previous sections, into a hypergraph. Finally, we provide the implementation of the Viterbi search algorithm for each of the decoders.

Huang and Chiang (2005) define a weighted directed hypergraph as follows:

[6] Contrary to Huang and Chiang (2007) we use probabilities instead of log scores in the computation of the item combinations.

[7] 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.

Definition 1 *An ordered hypergraph H is a tuple $\langle V, E, t, \boldsymbol{R} \rangle$, where V is a finite set of nodes, E is a finite set of hyperarcs and $\boldsymbol{R}$ is the set of weights. Each hyperarc $e \in E$ is a triple $e = \langle T(e), h(e), f(e) \rangle$, where $h(e) \in V$ is its head node, $T(e) \in V^*$ is a set of tail nodes and $f(e)$ is a monotonic weight function $\boldsymbol{R}_{|T(e)|}$ to $\boldsymbol{R}$ and $t \in V$ is a target node.*

Definition 2 *We impose the arity of a hyperarc to be $|e| = |T(e)| = 2$, in other words, each head node is connected with at most two tail nodes.*

Definition 3 *The backward-star BS(v) of a node v is the set of incoming hyperarcs $\{e \in E \,|\, h(e) = v\}$. The in-degree of v is $|BS(v)|$.*

Definition 4 *A derivation D of a node v is recursively defined as follows:*

- *If $e \in BS(v)$ with $|e| = 0$, then $D = e$, is a derivation of v, with size $|D| = 1$, and weight $w(D) = f(e)()$.*
- *If $e \in BS(v)$ where $|e| > 0$ and D_i is a derivation of $T_i(e)$ for $1 \leq i \leq |e|$, then $D = < e, D_1 \ldots D_{|e|} >$ is a derivation of v, its size $|D| = 1 + \sum_{i=1}^{|e|} |D_i|$ and its weight $w(D) = f(e)\left(w(D_1), \ldots, w(D_{|e|})\right)$.*

Definition 5 *Let $D_k v$ be the k^{th} best derivation of v, and $\mathbf{D}(v)$ be the list of k-best derivations $D_1(v), \ldots, D_k(v)$.*

Definition 6 *A derivation with back-pointers $\hat{D}$ of v is a tuple $< e, \mathbf{j} >$ such that $e \in BS(v)$, and $\mathbf{j} \in \{1, 2, \ldots, k\}^{|e|}$.*

Klein and Manning (2001) describe an automatic procedure to convert a grammar in Chomsky normal form and an input example to a weighted directed hypergraph. Analogously, we can convert both the basic and the extended decoder, as follows (for simplicity we will illustrate the procedure for the basic decoder only):

- each node $[A, i, j]$ in the hypergraph corresponds to an $[A, i, j]$ item spanning words of the input with indices from i to j;
- inference rule (1) of the basic decoder in Figure 4.6, is mapped to the hyperarc $\langle (B, i, j), (A, i, j), f \rangle$, where $f = s \cdot s_1$;
- similarly, rule (2) is mapped to the hyperarc $\langle ((B, i, k), (C, k, j)), (A, i, j), f \rangle$, with $f = s \cdot s_1 \cdot s_2$;

- the axioms corresponding to the lexical rules G_{SURF} are trivially mapped to the hyperarc $\langle \varepsilon, (A, i, i+1), f \rangle$, with $f = s$ and ε being the empty symbol.

The hypergraph can be thus viewed as an instantiation of the weighted deduction system. Figure 4.13 shows an example of the (partial) hypergraph representation of our grammar and the database input of Figure 3.4.

Next, we need to define a search algorithm that finds the best derivation (we will refer to it as the 1-best Viterbi search algorithm) in the hypergraph, much as we did with the Viterbi search algorithm for the basic decoder in Figure 4.7. Recall, that the basic decoder defines a specific order of combining items with smaller spans before items with larger spans, as well as antecedent items before consequent items. In order to do the same here we first need to traverse the hypergraph in a particular order:

Definition 7 *The graph projection of a hypergraph $H = \langle V, E, t, \mathbf{R} \rangle$ is a directed graph $G = \langle V, E' \rangle$ where $E' = \{(u, v) \mid \exists e \in BS(e), u \in T(e)\}$. A hypergraph H is considered to be acyclic if the graph projection G is a directed acyclic graph. Therefore, a topological ordering of H is an ordering of nodes V, which is also a topological ordering in G (from sources to target).*

Now we are ready to define the 1-best Viterbi search algorithm for a hypergaph H; the pseudocode is shown in Figure 4.14. The corner-stone step of the algorithm is to visit all nodes $v \in V$ in topological order. Then for each incoming hyperarc e of node v (i.e., the back-star $BS(v)$), we only need to update the 1-best derivation $\hat{D}_1$ list of v with the best scoring tuple $\langle e, \mathbf{1} \rangle$. The runtime complexity of the algorithm is $O(|E|)$, since the arity of the hypergraph is constant. A partial execution of this algorithm, is given in the example of Figure 4.13, which highlights the 1-best derivation path in red. Notice that the back-star of node $\mathrm{FS}_{0,2}(\mathrm{skyCover}_1, \mathrm{start})$ has 4 different hyperarcs, as a result of instantiating rules (4) and (5) of G_{CS} ($FS(r, r.f_i) \rightarrow F(r, r.f_j)\ FS(r, r.f_j)$ and $FS(r, r.f_i) \rightarrow F(r, r.f_j)$ respectively), for different value of fields f and spans:

$$BS\left(\mathrm{FS}_{0,2}(\mathrm{skyCover}_1, \mathrm{start})\right) = \left\{ \begin{array}{c} \mathrm{F}_{0,1}(\mathrm{skyCover}_1, \%)\ \mathrm{FS}_{1,2}(\mathrm{skyCover}_1, \mathrm{start}) \\ \mathrm{F}_{0,2}(\mathrm{skyCover}_1, \%) \\ \mathrm{F}_{0,1}(\mathrm{skyCover}_1, \mathrm{time})\ \mathrm{FS}_{1,2}(\mathrm{skyCover}_1, \mathrm{start}) \\ \mathrm{F}_{0,2}(\mathrm{skyCover}_1, \mathrm{time}) \end{array} \right\}$$

However, following the argumentation of Section 4.3.2, the best derivation returned by the 1-best Viterbi algorithm does not correspond to the best generated text. Hence

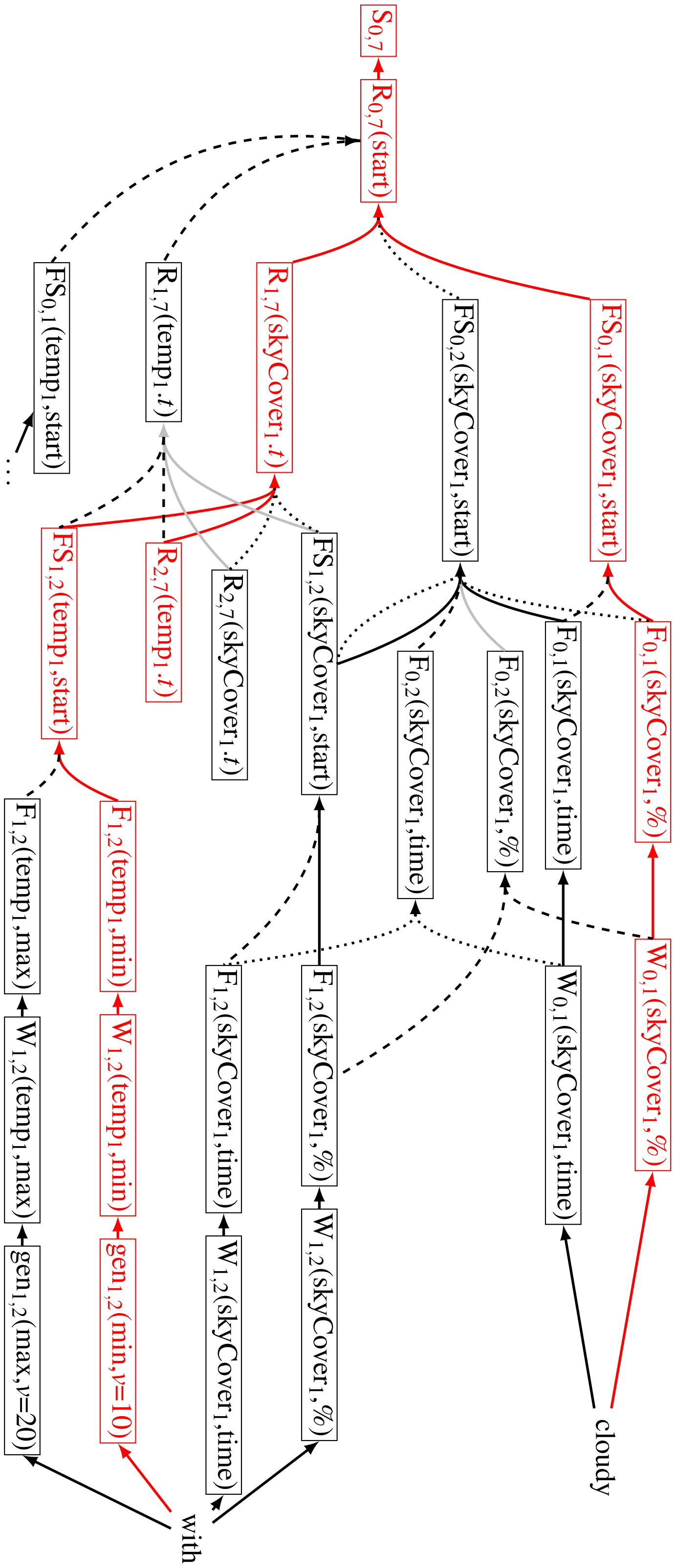

$S_{0,7}$
$R_{0,7}$(start)
$FS_{0,1}$($temp_1$,start)
$R_{1,7}$($temp_1$.t)
$R_{1,7}$($skyCover_1$.t)
$FS_{0,2}$($skyCover_1$,start)
$FS_{0,1}$($skyCover_1$,start)
$FS_{1,2}$($temp_1$,start)
$R_{2,7}$($temp_1$.t)
$R_{2,7}$($skyCover_1$.t)
$FS_{1,2}$($skyCover_1$,start)
$F_{0,2}$($skyCover_1$,time)
$F_{0,2}$($skyCover_1$,%)
$F_{0,1}$($skyCover_1$,time)
$F_{0,1}$($skyCover_1$,%)
$F_{1,2}$($temp_1$,max)
$F_{1,2}$($temp_1$,min)
$F_{1,2}$($skyCover_1$,time)
$F_{1,2}$($skyCover_1$,%)
$W_{0,1}$($skyCover_1$,time)
$W_{0,1}$($skyCover_1$,%)
$W_{1,2}$($temp_1$,max)
$W_{1,2}$($temp_1$,min)
$W_{1,2}$($skyCover_1$,time)
$W_{1,2}$($skyCover_1$,%)
$gen_{1,2}$(max,v=20)
$gen_{1,2}$(min,v=10)
cloudy
with
...

Figure 4.13: Partial hypergraph representation for the sentence *Cloudy with temperatures between 10 and 20 degrees.* For the sake of readability, we show a partial span on the first two words without weights on the hyperarcs. In red is the 1-best derivation path extracted via the Viterbi search algorithm of Figure 4.14.

```
procedure VITERBI($\langle V, E, t, \mathbf{R} \rangle$)
    for $v \in V$ in topological order do
        for $e \in BS(v)$ do
            $\hat{D}_1(v) \leftarrow max\left(\hat{D}_1(v), \langle e, \mathbf{1} \rangle\right)$
        end for
    end for
end procedure
```

Figure 4.14: 1-best Viterbi Search procedure on hypergraphs

we can extend it to search for the top k derivations $\hat{D}$ for each node v, using the lazy algorithm of Huang and Chiang (2005). The pseudocode of Figure 4.15 (Huang and Chiang, 2007) implements the cube pruning heuristic described in Section 4.3.2.4, directly on the hypergraph framework. The process is analogous to the approximate Viterbi search for the extended decoder, described in Section 4.3.2.4.

The notation $\langle e, \mathbf{j} \rangle$ identifies the derivation of v via the hyperedge e and the j_i^{th}-best sub-derivation of antecedent item u_i, where $1 \leq i \leq |j|$. $\mathbf{1}$ is a vector with all elements set to 1, $\mathbf{b}^i$ is a vector with all elements set to 0 except for the i^{th} which is set to 1. Function CUBE returns the top scoring derivation, by calling KBEST for each node v in the hypergraph in topological order. The procedure KBEST begins by initialising the priority queue *cand* with the top-left corner (recall Figure 4.12a) item from each hyperedge (lines 8–10). Then it explores the rest of derivations from the top-left corner and on (Figures 4.12b-c), by appending items out of order in a temporary buffer called *buf* (lines 11–15). Once the buffer is filled with k items (or there are no more left to explore), we sort it and store it to $\mathbf{D}(v)$, i.e., the list of derivations for the current node v under consideration. PUSHSUCC essentially enumerates all the combinations of the derivations lists of antecedent items in the correct order, as described in Section 4.3.2.4, i.e., pushes the successors $\{\langle e, \mathbf{j} + \mathbf{b}^i \rangle \; | i \in 1 \ldots |e|\}$ of node v along hyperedge e into *cand* (lines 20–25).

4.4 Experiments

Generation in our model amounts to finding the best derivation $(\hat{g}, \hat{h})$ that maximizes the product of two likelihoods, namely $p(g, h \,|\, \mathbf{d})$ and $p(g)$ (see equation (4.5)). $p(g, h \,|\, \mathbf{d})$ corresponds to the rules of G_{GEN} that generate the word sequence g, whereas $p(g)$ is

```
function CubePruning($\langle V, E, t, \mathbf{R} \rangle$)
    for $v \in V$ in topological order do
        KBest($v$)
    end for
    return $D_1(TOP)$
end function
procedure KBest(v)
    $cand \leftarrow \{\langle e, \mathbf{1} \rangle \mid e \in IN(v)\}$        ▷ for each incoming $e$
    Heapify($cand$)        ▷ a priority queue of candidates
    $buf \leftarrow \emptyset$
    while $|cand| > 0$ and $|buf| < k$ do
        $item \leftarrow$ Pop-Max($cand$)
        $buf \leftarrow item$
        PushSucc($item$, $cand$)
    end while
    sort $buf$ to $\mathbf{D}(v)$
end procedure
procedure PushSucc($\langle e, \mathbf{j} \rangle$, $cand$)
    $e$ is $u \rightarrow u_1 \dots u_{|e|}$
    for $i$ in $1 \dots |e|$ do
        $\mathbf{j}' \leftarrow \mathbf{j} + \mathbf{b}^i$
        if $|\mathbf{D}(u_i)| \geq j'_i$ then
            Push($\langle e, \mathbf{j}' \rangle, cand$)
        end if
    end for
end procedure
```

Figure 4.15: Cube pruning on hypergraphs (Huang and Chiang, 2007)

Dataset	Precision	Recall	F_1
RoboCup	86.4	86.4	86.4
WeatherGov	67.8	67.2	67.5
Atis	100	83.8	91.2
WinHelp	82.3	83.6	83.0

Table 4.3: Precision, Recall and F_1 results on the alignment task on RoboCup, WeatherGov, Atis, WinHelp.

the likelihood of g independently of $\mathbf{d}$. In the following sections we present how we trained the weights of our grammar G_{GEN}, how we estimate the hyperparameters k and β_{LM} of the model, as well as how we determine output text length N for each scenario before decoding.

4.4.1 Training G_{GEN} weights

In order to learn the weights of the grammar rules we directly estimate them on the hypergraph representation using the EM algorithm. Given a training set of scenarios with database records $\mathbf{d}$ and text w we maximize the marginal likelihood of the data, while summing out record tokens $\mathbf{r}$ and their fields $r_i.\mathbf{f}$, which can be regarded as latent variables:

$$\max_{\theta} \prod_{(w,\mathbf{d})} \sum_{\mathbf{r},\mathbf{f}} p(\mathbf{r},\mathbf{f},w|\mathbf{d};\theta), \tag{4.16}$$

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 a dynamic program similar to the inside-outside algorithm (Li and 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. Examples of the top scoring items of the multinomial distributions for some of the grammar rules of G_{GEN} are given in Table 4.4. For each domain we iterate EM until the F_1 score on the alignment task (see Section 3.3.1) stops increasing. In the case of WinHelp, the F_1 score was really low, possibly due to the limited number of training documents (128 scenarios). In order to overcome this, we adopted a form of staged learning: we ran first a set of EM iterations on a version of the corpus where each document (of the corresponding scenario) is split into sentences (recall that scenarios in the original dataset of Branavan et al. (2009) were split into sentences rather than documents), and then another set of iterations on the complete scenarios. By the first set of iterations we essentially provided the model with a good initial estimate of the rule weights, since we restrict considerably the choices of the model at the record level. According to Table 3.2 there are on average 9.2 records per document, or 2.1 per sentence, hence the content selection task on the sentence level is much easier. Then we relax this restriction by training on the whole document, using the rule weights obtained from the previous run on EM, and achieve a reliable F_1 score. Table 4.3 summarises the results on the alignment task, as described in Section 3.3.1, across all four domains. The results provide an interesting indication of the quality of the grammar rule weights obtained, since we use the same grammar to generate alignments as well. We can thus correlate better alignment performance with better trained rule weights. Recall, however, that the metrics used, measure alignment at the record level only, and not of the entire latent structure between records-fields-values and words. Also note that content selection in ATIS and WINHELP entails selecting all records in the database as they need to be specifically mentioned in the text, hence the relatively high scores in Table 4.3. WEATHERGOV on the other hand poses a different challenge as far as content selection is concerned; it has the highest number of records present in each scenario, out of which only 16.1% is mentioned (5.8 out of 36); this explains the comparatively low scores in the table, and will also introduce noise to the grammar weights.

4.4.2 Training external models

We obtain an estimate for $p(g)$ by linearly interpolating the score of a language model and DMV (Klein and Manning, 2004). Specifically, our language models were trained with the SRI toolkit Stolcke (2002) using add-1 smoothing.[8] 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[9] for each of our domains. All datasets were tagged automatically using the Stanford POS tagger (Toutanova et al., 2003) and words

[8] 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.

[9] When trained on the WSJ-10 corpus, our implementation of the DMV obtained the same accuracy as reported in Klein and Manning (2004). WSJ-10 consists of 7,422 sentences with at most 10 words after removing punctuation.

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

(a) RoboCup

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

(b) WeatherGov

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

(c) Atis

Table 4.4: 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 4.1, respectively). On the first column of each table is the underlying multinomial distribution for the corresponding rule.

k-BEST-LM	k
ROBOCUP	25
WEATHERGOV	15
ATIS	40
WINHELP	120

(a) Interpolation with LM

k-BEST-LM-DMV	k	β_{LM}
ROBOCUP	85	0.9
WEATHERGOV	65	0.3
ATIS	40	0.6
WINHELP	120	0.9

(b) Interpolation with LM and DMV

Table 4.5: 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.

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. We initialized EM to uniform distributions where a small amount of noise[10] 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 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 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 4.5.

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

[10] Repeated runs with different random noise on the WSJ-10 corpus yielded the same results; accuracy stabilized around the 60th iteration (out of 100).

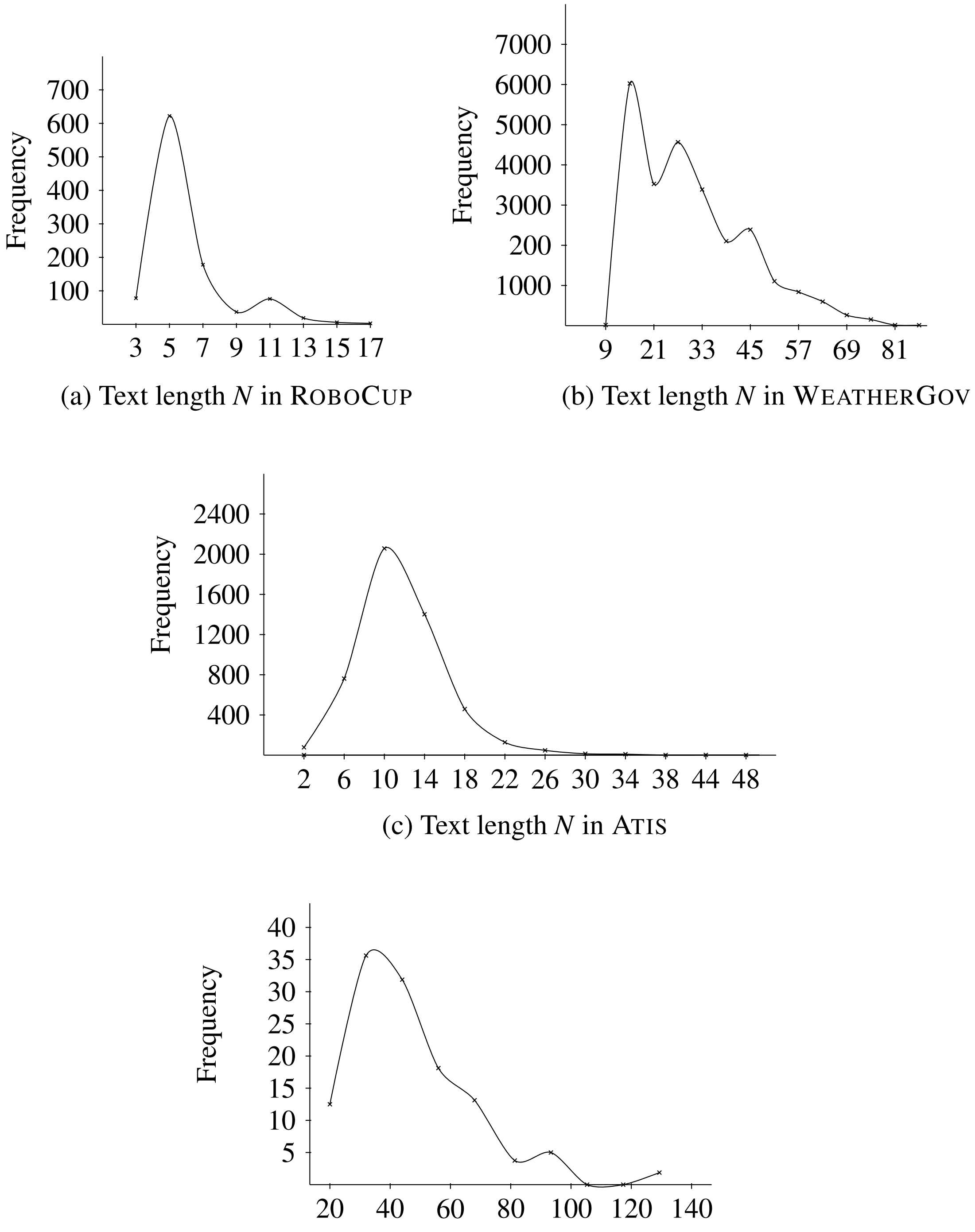
the DMV are integrated (k-BEST-LM-DMV). As can be seen, optimal values for k are generally larger for k-BEST-LM-DMV. This is probably due to noise introduced by the DMV; as a result, the decoder has to explore the search space more thoroughly. In an effort to investigate the impact of the DMV further, we fixed $\beta_{LM} = 0$ 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.

4.4.3 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 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 et al., 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 4.16 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 shorter. 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 (string, categorical or numerical) or its frequency in the training data. The former scheme worked better in denser datasets, such as WEATHERGOV WINHELP, 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, 0.73 for ATIS and 0.91 for WINHELP (using Pearson's r)[11].

[11] Note that the correlation coefficient for ROBOCUP is much lower than the rest. Error analysis on the results revealed that many errors were introduced while rounding off real numbers to integers. This can be further justified given that the average length (Figure 4.16) is sharply peaked around 5 words, hence



(a) Text length N in RoboCup

(b) Text length N in WeatherGov

(c) Text length N in Atis

(d) Text length N in WinHelp

Figure 4.16: Text length distribution in RoboCup, WeatherGov, Atis and WinHelp (training set).

4.4.4 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 4.4.1) and we omit them here for the sake of brevity. We compared the output of our models to Angeli et al. (2010) whose approach is closest to ours and state-of-the-art on the WEATHERGOV domain.[12] We trained their system on ATIS, and WINHELP, using the most reasonable parameter settings on their model after personal communication with the authors. We did not introduce further heuristics for the template extraction process as they report for ROBOCUP and WEATHERGOV as it is less straightforward to provide patterns for spontaneous speech (ATIS) and longer documents with often unique content in each document (WINHELP— names of objects, e.g., *activex controls* may be accounted for only once in the whole dataset). For ROBOCUP, we also compared against the best-published results (Kim and Mooney, 2010).

For the human evaluation study we compare two configurations of our systems, i.e., 1-BEST and *k*-BEST-LM-DMV along with Angeli's system and the human text (HUMAN) as gold-standard. We randomly selected 12 documents from the test set for each domain and generated output with our models. We thus obtained ratings for 48 (12 × 4) scenario-text pairs for each domain, from a total of 385 volunteers (104 for ROBOCUP, 101 for WEATHERGOV, 100 for ATIS, and 80 for WINHELP). For WINHELP, we made sure participants were computer-literate and familiar with the Windows operating system by administering a short questionnaire prior to the experiment. Our experimental instructions are given in Appendix C.

4.5 Results

In this section we show the results of our experiments on all four datasets, ROBOCUP, WEATHERGOV, ATIS and WINHELP, along with a discussion on example output.

a small deviation due to a truncation error, can decrease the accuracy of the regressor considerably compared to the rest datasets. WEATHERGOV also scores lower than the rest two datasets, however looking at the average length distribution reveals a rather irregular pattern, probably not captured in the best (though satisfactory) way via a linear regression model.

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

4.5.1 Automatic Evaluation

System	BLEU	METEOR
1-BEST	$8.01^{\triangleright\diamond}$	$34.29^{\triangleright\diamond}$
k-BEST-LM	24.88^{*}	52.22^{*}
k-BEST-LM-DMV	23.14^{*}	$46.90^{*\triangleright}$

(a) Joint Content Selection

System	BLEU	METEOR
1-BEST	$10.79^{\triangleright\diamond\circ\dagger}$	$27.70^{\triangleright\diamond\circ\dagger}$
k-BEST-LM	30.90^{*}	57.77^{*}
k-BEST-LM-DMV	29.73^{*}	58.05^{*}
ANGELI	28.70^{*}	–
KIM-MOONEY	$47.27^{*\triangleright\diamond\circ}$	–

(b) Fixed Content Selection

Table 4.6: BLEU-4 and METEOR scores on ROBOCUP (*: 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 and Mooney (2010)).

ROBOCUP **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 Table 4.6a. 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 and Mooney, 2010).[13] Our results are summarized in Table 4.6b.

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 and METEOR scores among k-BEST-LM-DMV, k-BEST-LM and ANGELI are not statistically signif-

[13] 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.

System	BLEU	METEOR
1-BEST	$8.64^{\triangleright\diamond\circ}$	$16.35^{\triangleright\diamond\circ}$
k-BEST-LM	$33.70^{*\diamond\circ}$	$51.47^{*\circ}$
k-BEST-LM-DMV	$34.18^{*\triangleright\circ}$	$52.25^{*\triangleright\circ}$
ANGELI	$38.40^{*\triangleright\diamond}$	$60.50^{*\triangleright\diamond}$

Table 4.7: BLEU-4 and METEOR scores WEATHERGOV (*: 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).

icant (except in the case of joint content selection, where the difference in METEOR between *k*-BEST-LM and *k*-BEST-LM-DMV is 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 considerably 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.

WEATHERGOV **Results** 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 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.[14] 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.

[14] 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).

System	BLEU	METEOR
1-BEST	$11.85^{\triangleright\diamond\circ}$	$32.77^{\triangleright\diamond\circ}$
k-BEST-LM	$29.30^{*\diamond}$	$43.70^{*\diamond}$
k-BEST-LM-DMV	$30.37^{*\circ}$	$45.40^{*\circ}$
ANGELI	$26.77^{*\diamond}$	$42.41^{*\triangleright\diamond}$

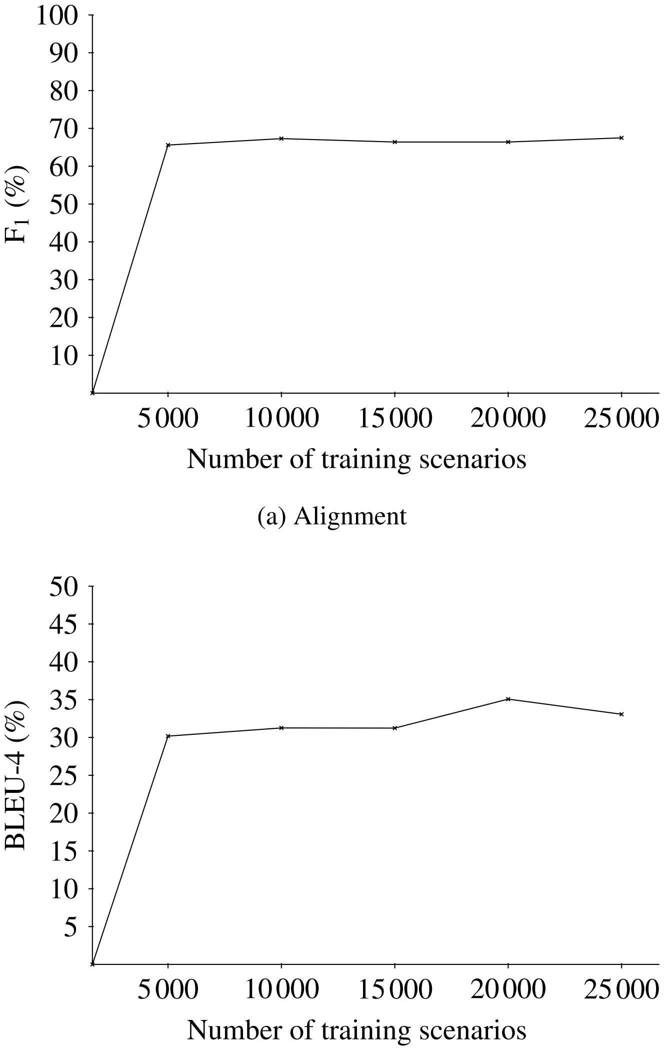
Table 4.8: BLEU-4 and METEOR scores on ATIS (*: significantly different from 1-BEST; $^\circ$: significantly different from ANGELI; $^\triangleright$ significantly different from *k*-BEST-LM; $^\diamond$: significantly different from *k*-BEST-LM-DMV).

System	BLEU	METEOR
1-BEST	$16.02^{\triangleright\diamond\circ}$	$28.02^{\triangleright\diamond\circ}$
k-BEST-LM	$38.26^{*\diamond}$	$51.32^{*\circ}$
k-BEST-LM-DMV	$39.03^{*\circ}$	$51.68^{*\circ}$
ANGELI	$32.21^{*\triangleright\diamond}$	$35.33^{*\triangleright\diamond}$

Table 4.9: BLEU-4 and METEOR scores on WINHELP (*: 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).

ATIS **Results** The results on ATIS are shown in Table 4.8. As we can see, 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 ($p < 0.01$). All the differences between the models are significant in METEOR. The ATIS domain is more challenging than the previous datasets 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.

WINHELP **Results** Table 4.9 shows the results on the WINHELP domain. Again we notice that *k*-BEST-LM-DMV outperforms significantly the 1-BEST ($p < 0.01$), ANGELI ($p < 0.01$) models in terms of BLEU and METEOR scores, and *k*-BEST-LM ($p < 0.01$) in terms of BLEU score. This dataset requires from the generator precise content selection, since the order of the records plays a crucial role in understanding the generated text. Notice for example in Figure 4.19 how the 1-BEST model obscures the understanding of the text, when it mentions *'Click start'* in the middle rather than in the beginning of the text.



(a) Alignment

(b) Generation output

Figure 4.17: Learning curves for WEATHERGOV displaying how the quality of the alignments and generated output vary as a function of the size of the training data.

Learning Curves 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 the rest of the domains and is more challenging with regard to content selection. Figures 4.17(a) and (b) show how the number of training instances influences the quality of the alignment and generation output, respectively. We measure F_1-score for the alignment task and BLEU-4 for the generation output. The

	RoboCup		WeatherGov	
System	F	SC	F	SC
1-best	$2.14^{\diamond\dagger\circ}$	$2.09^{\diamond\dagger\circ}$	$2.25^{\diamond\dagger\circ}$	$2.53^{\diamond\dagger\circ}$
k-best-lm-dmv	4.05^{*}	$3.55^{*\dagger}$	3.89^{*}	3.54^{*}
Angeli	4.01^{*}	$3.47^{*\dagger}$	3.82^{*}	3.72^{*}
Human	4.17^{*}	$3.97^{*\diamond\circ}$	4.01^{*}	3.58^{*}
	Atis		**WinHelp**	
System	F	SC	F	SC
1-best	$2.40^{\diamond\dagger\circ}$	$2.49^{\diamond\dagger\circ}$	$2.57^{\diamond\dagger\circ}$	$2.10^{\diamond\dagger\circ}$
k-best-lm-dmv	3.96^{*}	$3.82^{*\circ}$	$3.41^{*\dagger}$	$3.05^{*\dagger}$
Angeli	3.86^{*}	$3.31^{*\dagger\diamond}$	$3.57^{*\dagger}$	$2.80^{*\dagger}$
Human	4.16^{*}	$3.96^{*\circ}$	$4.15^{*\diamond\circ}$	$4.04^{*\diamond\circ}$

Table 4.10: Mean ratings for fluency (F) and semantic correctness (SC) on system output elicited by humans on RoboCup, WeatherGov, Atis and WinHelp (*: significantly different from 1-best; $^{\circ}$: significantly different from Angeli; $^{\diamond}$: significantly different from *k*-best-lm-dmv; †: significantly different from Human).

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.

4.5.2 Human Evaluation Results

The results of our human evaluation study are shown in Table 4.10. 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 judgements only for *k*-best-lm-dmv as it generally performed better than *k*-best-lm in our automatic evaluation (see Tables 4.6 -4.9). 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 mean differences are statistically significant.

On all four 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) on ROBOCUP, WEATHERGOV and ATIS (pair-wise differences between k-BEST-LM-DMV, and HUMAN are not statistically significant) but not on WINHELP. Our output is also indistinguishable compared to ANGELI in all four domains. 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. Finally, on WINHELP the difference on semantic correctness between k-BEST-LM-DMV and ANGELI is not significant, even though our model scores higher; both are also significantly worse than HUMAN.

4.5.3 System Output

Examples of system output with correct content selection at the record level are given in Figures 4.18-4.19. 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 4.20a shows examples with incorrect content selection at the record level for the WEATHERGOV domain. Figure 4.20a shows the gold standard content selection and its corresponding verbalization. Figures 4.20b and 4.20c 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 4.21 shows the dependency structure our model produced for the sentence *Show me the flights from Milwaukee*

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**
06-21	32	39	46

Cloud Sky Cover	
Time	**Percent (%)**
06-21	75-100

Wind Speed			
Time	**Min**	**Mean**	**Max**
06-21	6	7	10

Wind Direction	
Time	**Mode**
06-21	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
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 4.18: Example output on (a) sportscasting, (b) weather forecasting, and (c) air travel domains with correct content selection.

Input:

navigate-desktop

envCmd	objName	type
left click	start	button

navigate-start

envCmd	objName	type
left click	settings	button

navigate-start-target

envCmd	objName	type
left click	control panel	button

navigate-window-target

envCmd	objName	type
double click	system	item

navigate-window-target

envCmd	objName	type
left click	hardware	tab

action-contextMenu

envCmd	objName	type	typeInto
left click	device manager	button	–

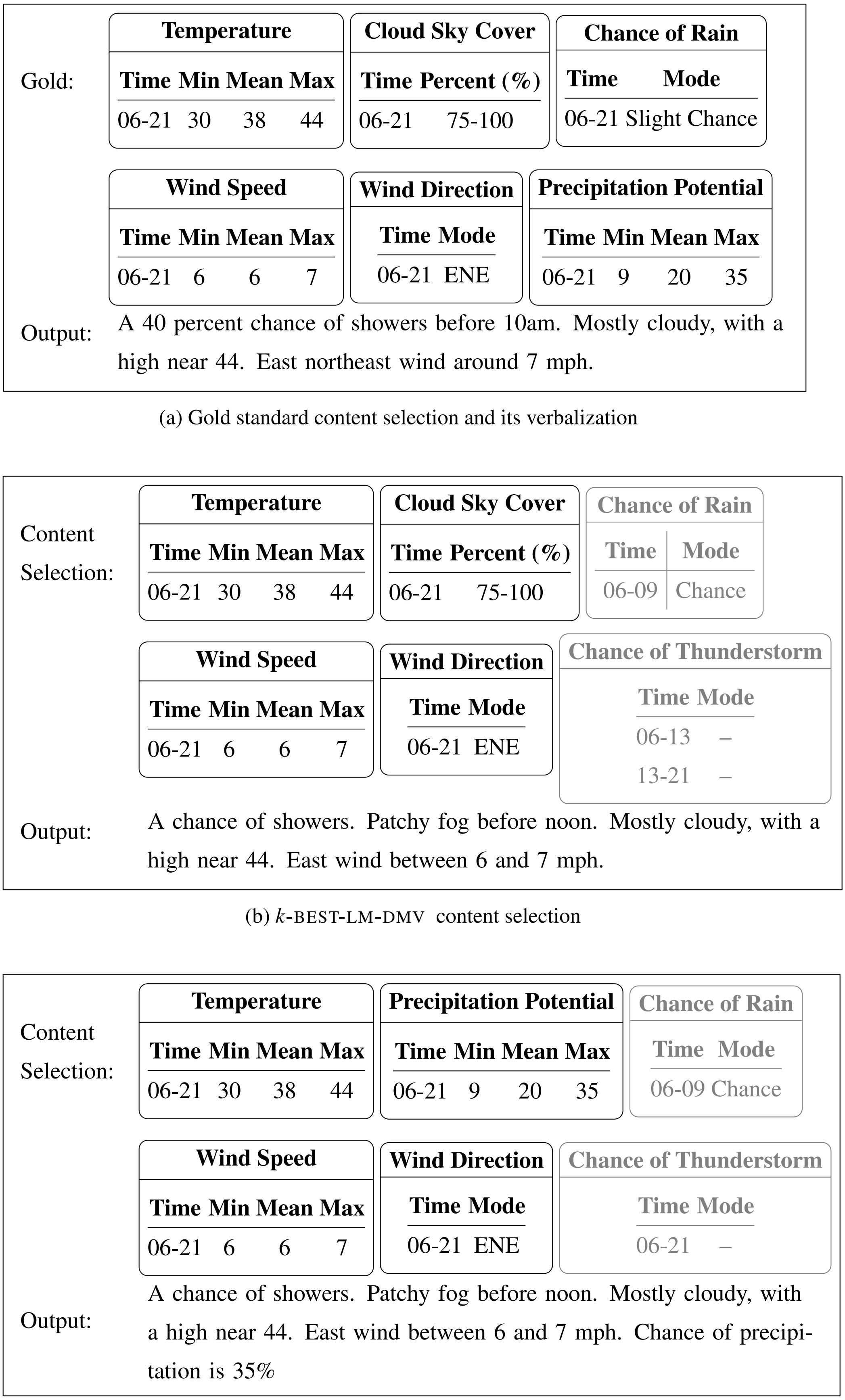
1-BEST: Select settings, click control panel. Click control panel. Double-click system. Click start, click device manager.

k-BEST-LM-DMV: Click start, point to settings, and then click control panel. Double-click system. Click the hardware tab, click device manager.

ANGELI: Click start , point to settings, and then click control panel. Double-click system. Click modems, and then click hardware.

HUMAN: Click start, point to settings, click control panel, and then double-click system. On the hardware tab, click device manager.

Figure 4.19: Example output on the troubleshooting guide domain with correct content selection.



Gold:

Temperature

Time	Min	Mean	Max
06-21	30	38	44

Cloud Sky Cover

Time	Percent (%)
06-21	75-100

Chance of Rain

Time	Mode
06-21	Slight Chance

Wind Speed

Time	Min	Mean	Max
06-21	6	6	7

Wind Direction

Time	Mode
06-21	ENE

Precipitation Potential

Time	Min	Mean	Max
06-21	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
06-21	30	38	44

Cloud Sky Cover

Time	Percent (%)
06-21	75-100

Chance of Rain

Time	Mode
06-09	Chance

Wind Speed

Time	Min	Mean	Max
06-21	6	6	7

Wind Direction

Time	Mode
06-21	ENE

Chance of Thunderstorm

Time	Mode
06-13	–
13-21	–

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
06-21	30	38	44

Precipitation Potential

Time	Min	Mean	Max
06-21	9	20	35

Chance of Rain

Time	Mode
06-09	Chance

Wind Speed

Time	Min	Mean	Max
06-21	6	6	7

Wind Direction

Time	Mode
06-21	ENE

Chance of Thunderstorm

Time	Mode
06-21	–

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

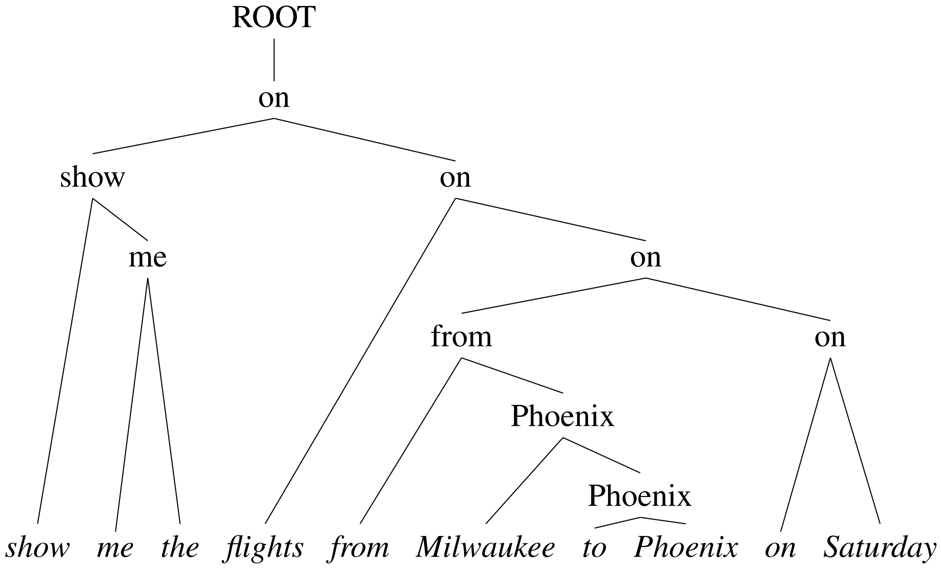


Figure 4.21: Dependency structure for the sentence *Show me the flights from Milwaukee to Phoenix on Sunday* as generated by *k*-BEST-LM-DMV (see Figure 4.18c). Intermediate nodes in the tree denote the head words of each subtree.

to Phoenix on Saturday from Figure 4.18c; notice the long range dependency between *flights* and *on*, which would otherwise be inaccessible to a language model.

4.6 Discussion

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 and WINHELP when evaluated both automatically and by humans (on ATIS). 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. In the WINHELP domain what is suggested by the generated output, is a frequent inconsistency between the acquired lexicon and the values of the fields of the database records. This is also illustrated in Figure 4.19; notice how the value `hardware` is lexicalised as *'modem'*.

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 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 information about heads and their dependants 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, i.e., the size of the list of derivations we need to keep at each node. Note, that fine-tuning k becomes less important when integrating with a language model only. As we explain in Section 4.4.1, 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.

Obtaining rule weights for our model in an unsupervised fashion using EM, is not of course the only way to train our grammar. We could use a supervised approach such as a log-linear model to train our model either in a fully discriminative setting (Chiang, 2007) or during the M-step similar to Berg-Kirkpatrick et al. (2010). In both cases however, we would need some form of supervision either in the form of alignments or features. For the former we can directly use the gold-standard or we may obtain them using a supervised alignment model (Snyder and Barzilay, 2007). For the features we can use knowledge directly from the text, such as text length, number of sentences, syntactic features, as well as from the database, such as patterns of records, and fields. Increasing the performance of the generation of alignments via better training of the rule weights of our model, will inevitably increase the quality of the generated output, hence the overall performance of our generator. Recall though that most domains do not contain annotation in the form of alignments; obtaining them manually in order to train a supervised model can be expensive. Creating in-domain features based on evidence from the document or the database, still requires human intervention. Therefore we regard the tradeoff between unsupervised training of our model and performance as a reasonable solution especially for domains with no annotation.

4.7 Summary

In this chapter we presented a joint model for content selection, sentence planning, and surface realisation. The key idea was to recast generation as a parsing problem. We achieved this by introducing a PCFG grammar that naturally captures the correspondence between the database schema and the text. Beginning at the root non-terminal symbol, we first defined a set of rewrite rules that describe a sequence of records; then for each record we defined another a similar set of rules that emit a sequence of fields. Finally, for each field value we included a set of lexicalisation rules that emit words. The grammar is purely syntactic and does not relate to a specific domain. We then proposed several models to parse an input database and our PCFG in order to generate text, via integration with a language model and a dependency model. We also provided an implementation of our generators using the hypergraph framework. Finally, we concluded with an extensive evaluation of our systems across the four domains presented in Chapter 3. We achieved performance comparable or superior compared to the state-of-the-art system of Angeli et al. (2010). However, our models lag behind the system of Kim and Mooney (2010) on ROBOCUP, possibly due to the latter relying on more supervision during training and decoding. The output of our models was semantically compatible and often indistinguishable from the gold-standard text, according to human judges. More importantly our models learnt domain specific decisions on "what to say" and "how to say" from data, without any hand-engineering or manual annotation. Given a database input and collocated text, we argue that our systems can easily port to a different domain.

Chapter 5

Integrating Document Planning

The grammar we defined in the previous chapter captures both the structure of the input database and the way it renders into natural language. This approach lends itself well to the incorporation of content planning, which has traditionally assumed tree-like representations. In this chapter, we will look at augmenting the original grammar G_{GEN} with an additional context-free sub-grammar G_{DP} which performs document planning. We formulate G_{DP} so that it identifies sentences in a document, and captures the inter- and intra-relations of records in them. Importantly, we do not specify this grammar manually but obtain it automatically from training data. In the following we present the extensions on the original grammar, a procedure for extracting rules for G_{DP}, and a modified training scheme. We evaluate the extended model on two datasets, namely WEATHERGOV and WINHELP.

5.1 Motivation

Content planning is a fundamental component in a natural generation system. Not only does it determine which information-bearing units to talk about, but also arranges them into a structure that creates coherent output. It is therefore not surprising that many content planners, as we saw earlier in chapter 2, have been based on theories of discourse coherence (Hovy, 1993; Scott and de Souza, 1990). Other work has relied on generic planners (Dale, 1988) or schemas (Duboue and McKeown, 2002). In all cases, content plans are created manually, sometimes through corpus analysis. A few researchers recognize that this top-down approach to planning is too inflexible and adopt a generate-and-rank architecture instead (Mellish et al., 1998; Karamanis, 2003; Kibble and Power, 2004). The idea is to produce a large set of candidate plans and select the best one according to a ranking function. The latter is typically developed manually taking into account constraints relating to discourse coherence and the semantics of the domain.

The first statistical approaches to induce document plans are the work of Duboue and McKeown (2001) and Duboue and McKeown (2002), that model content planning using methods borrowed from computational biology, presented in Section 2.3.1. More recent data-driven work mentioned in Chapter 2 (Kim and Mooney, 2010; Angeli et al., 2010) focuses on end-to-end systems rather than individual components, however without taking document planning into account.

The model we presented in the previous chapter optimises the choice of records, fields and words simultaneously; however, it still selects and orders records locally. In the following we will extend our model by replacing the existing content selection mechanism (based on a simple markovized chaining of records) with a more global representation of the document. A *document plan* is identified as a sequence of sentences, and each sentence contains a sequence of records. Unlike in the original model, the choice and ordering of records is performed globally, and is not merely based on the previous record choice.

5.2 Grammar Extensions

We propose replacing rules (1)–(2) from grammar G_{GEN} in Figure 4.1 with the following:

Definition 8 (G_{DP} grammar)

1. $D \rightarrow SENT(t_i,\, \ldots,\, t_j)\, \ldots\, SENT(t_l,\, \ldots,\, t_m)$
2. $SENT(t_i,\, \ldots,\, t_j) \rightarrow R(r_a.t_i)\, \ldots\, R(r_k.t_j)\, \cdot$

where t is a record type, t_i, t_j, t_l and t_m may overlap and r_a, r_k are record tokens of type t_i and t_j respectively. The corresponding weights are:

Definition 9 (G_{DP} weights)

1. $P(t_i,\, \ldots,\, t_j,\, \ldots\, t_l,\, \ldots, t_m \mid D)$
2. $P(t_i)\, \cdot \ldots\, \cdot P(t_j) = \frac{1}{|\mathbf{s}(t_i)|}\, \cdot\, \ldots\, \cdot\, \frac{1}{|\mathbf{s}(t_j)|}$

where $\mathbf{s}(t)$ is the function that returns the set of records with type t, defined in Section 4.1. The resulting grammar G_{GEN}++ is given in Table 5.1. The rules in Definition 8 essentially describe a document grammar on record types. We split a document

G_{DP}	1. $\mathrm{D} \rightarrow \mathrm{SENT}(t_i, \ldots, t_j) \ldots \mathrm{SENT}(t_l, \ldots, t_m)$	$\left[P(t_i, \ldots, t_j, \ldots t_l, \ldots, t_m \mid D)\right]$
	2. $\mathrm{SENT}(t_i, \ldots, t_j) \rightarrow \mathrm{R}(r_a.t_i) \ldots \mathrm{R}(r_k.t_j) \cdot$	$\left[\frac{1}{\lvert\mathbf{s}(t_i)\rvert} \cdot \ldots \cdot \frac{1}{\lvert\mathbf{s}(t_j)\rvert}\right]$
G_{CS}	3. $\mathrm{R}(r_i.t) \rightarrow \mathrm{FS}(r_j, start)$	$[Pr = 1]$
	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. $\mathrm{W}(r, r.f) \rightarrow \mathrm{gen}(f.v)$	$[P(\mathrm{gen}(f.v).mode \mid r, r.f, f.t = int) \cdot$ $P(f.v \mid gen(f.v).mode)]$
	10. $\mathrm{W}(r, r.f) \rightarrow gen\_str(f.v, i)$	$[Pr = 1]$

Table 5.1: Grammar rules for G_{GEN}++ and their weights shown in square brackets.

into sentences, each terminated by a full-stop. Then a sentence is further split into a sequence of record types. Contrary to the original model, we observe at the root node D a complete sequence[1] of record types, split into sentences. This way we aim to learn domain-specific patterns of frequently occurring record type sequences among the sentences of a document, as well as more local structures inside a sentence.

Rule (1) defines the expansion from the start symbol D to a sequence of sentences, each represented by the non-terminal $SENT$. Similarly to the original grammar G_{GEN}, we employ the use of features (shown in parentheses) to denote a sequence of record types. We assume that the same record types may recur in different sentences, but not in the same one. The weight of rule (2) is simply the joint probability of all the record types present, ordered and segmented appropriately into sentences in the document, given the start symbol.

[1] Note that a sequence is different from a permutation, as we may allow repetitions or omissions of certain record types.

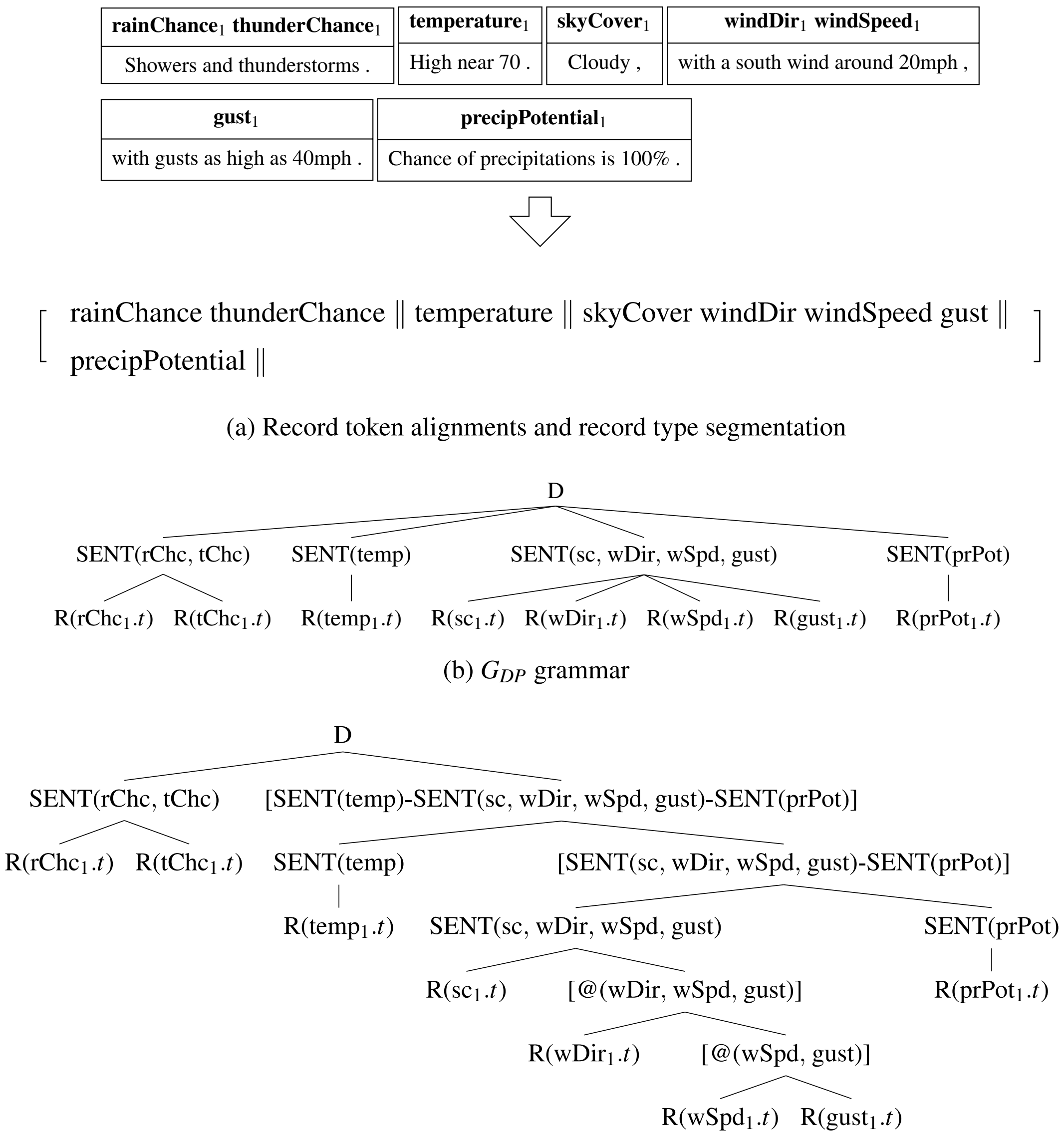
Once record types have been selected (on a per sentence basis) we move on to rule (2) which describes how each non-terminal *SENT* expands to an ordered sequence of records R, as they are observed within a sentence (note the terminal symbol '.' at the end of the rule). Notice that a record type t_i may correspond to several record tokens r_a. Rules (4)–(8) in grammar G make decisions on these tokens based on the overall content of the database and the field/value selection. The weight of this rule is the product of the weights of each record type. This is set to the uniform distribution over $\{1, ..., |\mathbf{s}(t)|\}$ for record type t, where $|\mathbf{s}(t)|$ is the number of records with that type.

Figure 5.2b shows an example tree of the database input of Figure 3.6, using the rules of G_{DP}, assuming that the alignments between records and text are given. The top level span refers to the sequence of record types as they are observed in the text. The first sentence contains three records with types 'desktop', 'start' and 'start-target', each corresponding to the textual segments *click start*, *point to settings*, *and then click control panel.* The next level on the tree denotes the choice of record tokens for each sentence, provided that we have decided on the choice and order of their types. In the example, the bottom-left sub-tree corresponds to the choice of the first three records of Figure 3.6.

5.3 Model Training

A straightforward way to train the extended model would be to embed the parameters of G_{DP} in the original model and then run the EM algorithm using inside-outside at the E-step. We would first need to binarize the two extra rules, in order to keep the cubic runtime bound of the algorithm (see Figures 5.1c, 5.2c). Unfortunately, this method will induce a prohibitively large search space. Rule (1) enumerates all possible combinations of record type sequences and the number grows exponentially even for a few record types and a small sequence size. To tackle this problem, we extracted rules for G_{DP} from the training data, based on the assumption that there will be far fewer unique sequences of record types per dataset than exhaustively enumerating all possibilities. Our extraction process proceeds as follows:

- For each scenario we obtain a word-by-word alignment between the database records and the corresponding text. In our experiments we adopted the method we used for the alignment task described in Section 3.3.1, similar to Liang et al.'s (2009) unsupervised model. However any other semi- or fully supervised

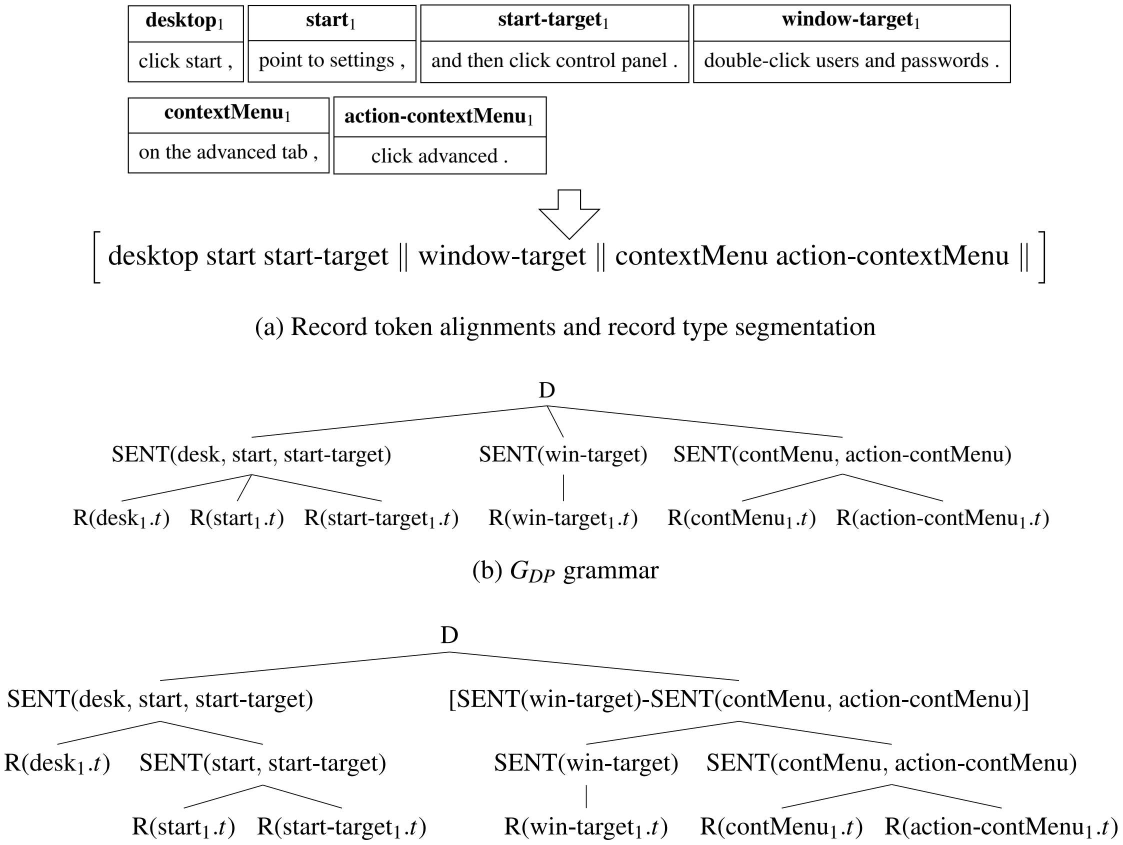

rainChance1 thunderChance1
Showers and thunderstorms .
temperature1
High near 70 .
skyCover1
Cloudy ,
windDir1 windSpeed1
with a south wind around 20mph ,
gust1
with gusts as high as 40mph .
precipPotential1
Chance of precipitations is 100% .
rainChance thunderChance || temperature || skyCover windDir windSpeed gust || precipPotential ||
D
SENT(rChc, tChc)
SENT(temp)
SENT(sc, wDir, wSpd, gust)
SENT(prPot)
R(rChc1.t)
R(tChc1.t)
R(temp1.t)
R(sc1.t)
R(wDir1.t)
R(wSpd1.t)
R(gust1.t)
R(prPot1.t)
D
SENT(rChc, tChc)
[SENT(temp)-SENT(sc, wDir, wSpd, gust)-SENT(prPot)]
R(rChc1.t)
R(tChc1.t)
SENT(temp)
[SENT(sc, wDir, wSpd, gust)-SENT(prPot)]
R(temp1.t)
SENT(sc, wDir, wSpd, gust)
SENT(prPot)
R(sc1.t)
[@(wDir, wSpd, gust)]
R(prPot1.t)
R(wDir1.t)
[@(wSpd, gust)]
R(wSpd1.t)
R(gust1.t)

(a) Record token alignments and record type segmentation

(b) G_{DP} grammar

(c) Binarized G_{DP} grammar

Figure 5.1: Grammar extraction example from the WEATHERGOV domain. We use *rChc* as a shorthand for the record type **Rain Chance**, *tChc* for **Thunder Chance**, *temp* for **Temperature**, *sc* for **Sky Cover**, *wDir* for **Wind Direction**, *wSpd* for **Wind Speed** and *prPot* for **Precipitation Potential**. (a) We first take the alignments of records on words, map them to their corresponding types and segment into sentences. (b) We next create a tree using grammar G_{DP} and (c) right-binarize it.


desktop$_1$
click start ,
start$_1$
point to settings ,
start-target$_1$
and then click control panel .
window-target$_1$
double-click users and passwords .
contextMenu$_1$
on the advanced tab ,
action-contextMenu$_1$
click advanced .
[desktop start start-target || window-target || contextMenu action-contextMenu ||]
D
SENT(desk, start, start-target)
SENT(win-target)
SENT(contMenu, action-contMenu)
R(desk$_1$.t)
R(start$_1$.t)
R(start-target$_1$.t)
R(win-target$_1$.t)
R(contMenu$_1$.t)
R(action-contMenu$_1$.t)
D
SENT(desk, start, start-target)
[SENT(win-target)-SENT(contMenu, action-contMenu)]
R(desk$_1$.t)
SENT(start, start-target)
SENT(win-target)
SENT(contMenu, action-contMenu)
R(start$_1$.t)
R(start-target$_1$.t)
R(win-target$_1$.t)
R(contMenu$_1$.t)
R(action-contMenu$_1$.t)

(a) Record token alignments and record type segmentation

(b) G_{DP} grammar

(c) Binarized G_{DP} grammar

Figure 5.2: Grammar extraction example from the WINHELP domain: (a) We first take the alignments of records on words, map them to their corresponding types and segment into sentences. (b) We next create a tree using grammar G_{DP} and (c) right-binarize it.

method could be used instead. As we show in Section 5.5, the quality of the alignment inevitably correlates with the quality of the extracted grammar and the decoder's output.

- Then we map the aligned record tokens to their corresponding types, merge adjacent words with the same type and segment on punctuation (see Figure 5.2a).

- Next, we create the corresponding tree according to the G_{DP} and binarize it (Figures 5.1b–5.1c, 5.2b–5.2c) We experimented both with left and right binarization and adhered to the latter, as it obtained a more compact set of rules.

- Finally, we collectively count the rule weights on the resulting treebank and extract a rule set, by keeping the rules with frequency greater than two.

Returning to the model, we run the EM algorithm via inside-outside using the extracted G_{DP} rules in order to build the hypergraph for each scenario, and learn the weights for the remaining rules (3–10), as described in Section 4.4.1. Decoding remains the same as described in the previous chapter; the only requirement is that the extracted grammar remains binarized in order to guarantee the cubic bound of the Viterbi search algorithm.

5.4 Experiments

Since our aim is to evaluate the planning component of the new model, we used datasets whose documents are at least a few sentences long, hence we experimented only on WEATHERGOV and WINHELP. In the following we describe the particulars of extracting the grammar rules for G_{DP} for the two domains and the evaluation methodology we followed, which departs slightly from the experiments described in the previous chapter.

5.4.1 Grammar Extraction

We obtained alignments between database records and textual segments for both domains using the original grammar G_{GEN} to perform alignment, as described in Section 3.3.1, similar to the unsupervised model of Liang et al. (2009). For WEATHERGOV, we then extracted 663 rules (after binarization), 344 of which were rooted on the start symbol D.

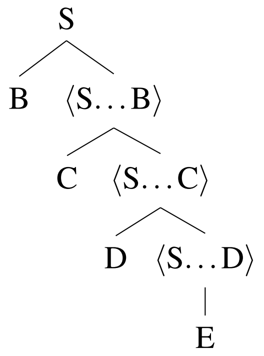


Figure 5.3: Horizontal markovisation of the rule $S \rightarrow B\ C\ D\ E$.

The WINHELP dataset is considerably smaller, and as a result following the procedure described above in Section 5.3 yields a very sparse grammar, with too many low-frequency rules. To alleviate this, we use the paradigm of Markov chains to decompose long rules to a chain of binary subrules. Based on Collins (1999) we horizontally *markovised* the right-hand side (RHS) of each grammar rule. We can encode an arbitrary amount of context in the intermediate non-terminals that result from this process; in our case we store h=1 horizontal siblings plus the mother left-hand side (LHS) non-terminal, in order to uniquely identify the Markov chain. For example, given a rule $S \rightarrow B\ C\ D\ E$ we can transform it into a first-order Markov model as shown in Figure 5.3. The probability score decomposes into: $P(B,C,D,E|S) = P(B|S)\ P(C|B,S)\ P(D|C,S)\ P(E|D,S)$. After markovisation, we obtained a grammar with 516 rules; 60 rules were rooted on D. Examples of extracted rules for both domains are given in Appendix B.

5.4.2 Training External Models

We estimated the two hyperparameters of the model, namely the number of k-best derivations considered by the decoder and β_{LM}, the vector of weights for integrating the language model and DMV, by performing grid search on held-out data of the development set for each domain. Table 5.2 shows the optimal values for k and β_{LM}. DP-UNSUP and DP-AUTO are two different configurations of our model defined in Section 5.4.3.

Dataset	k	β_{LM}
WeatherGov	75	0.9
WinHelp	120	0.5

(a) DP-Unsup

Dataset	k	β_{LM}
WeatherGov	80	0.3
WinHelp	120	0.5

(b) DP-Auto

Table 5.2: Optimal values for parameters k and β_{LM} calculated by performing grid search against BLEU-4 on the development set, using the G_{GEN}++ grammar. The two tables correspond to the different configurations of our model, one with extracted rules for the sub-grammar G_{DP} from unsupervised alignments, and the other from automatically extracted alignments using hand-crafted heuristics, respectively.

5.4.3 System Comparison

We evaluated two configurations of the new model, both of which integrate with a language model and the DMV. The first configuration DP-Unsup uses our grammar extracted from the unsupervised alignments. In the second configuration, the grammar is extracted from better quality alignments, the latter also obtained automatically but relying on human-crafted alignment heuristics which are based on domain knowledge, as described in Sections 3.2.2 and 3.2.4 (DP-Auto). As a baseline, we used the model presented in the previous chapter using grammar G_{GEN}. We also compared our model to Angeli et al.'s system (2010).

Our evaluation methodology is the same as in Chapter 4; we use BLEU and METEOR to automatically evaluate the output of our system, and elicited a human evaluation study. Since our aim is to measure the impact of the document planning component on the new model, we extended our human evaluation study by adding a third dimension besides fluency and semantic correctness. Participants were asked to rate our output in terms of coherence (is the text comprehensible and logically structured?). We used again a five point rating scale for coherence, with high values corresponding to better performance. Our experimental instructions are given in Appendix C. We compared the new model DP-Unsup with the extracted grammar from unsupervised data, and the k-best-lm-dmv using the original grammar G_{GEN} against Angeli et al. (2010) and the human text. We obtained ratings for 48 (12 $\times$ 4) scenario-text pairs for the two domains, from a total of 160 volunteers (80 for WeatherGov, and 80 for WinHelp).

	WeatherGov		WinHelp	
System	BLEU	METEOR	BLEU	METEOR
DP-Unsup	$36.41^{*\circ\dagger}$	$55.73^{*\circ\dagger}$	$41.73^{*\circ}$	$54.74^{*\circ}$
DP-Auto	$39.00^{*\circ\diamond}$	$58.55^{*\circ\diamond}$	$42.69^{*\circ}$	$55.45^{*\circ}$
k-best-lm-dmv	$34.18^{\circ\diamond\dagger}$	$52.25^{\circ\diamond\dagger}$	$39.03^{\circ\diamond\dagger}$	$51.68^{\circ\diamond\dagger}$
Angeli	$38.40^{*\diamond\dagger}$	$60.50^{*\diamond\dagger}$	$32.21^{*\diamond\dagger}$	$35.33^{*\diamond\dagger}$

Table 5.3: Automatic evaluation of system output using BLEU-4 and METEOR. (*: significantly different from *k*-best-lm-dmv; $^\circ$: significantly different from Angeli; $^\diamond$: significantly different from DP-Unsup; †: significantly different from DP-Auto).

5.5 Results

The results of the automatic evaluation are summarized in Table 5.3. Overall, the new models outperform the baseline *k*-best-lm-dmv by a wide margin on both datasets. DP-Auto is superior to DP-Unsup (on WeatherGov the difference is statistically significant) and Angeli on WinHelp, while on WeatherGov in terms of BLEU only. This is not entirely unexpected, given the better quality of the extracted rules. On WeatherGov this difference is more pronounced. This is probably because the dataset shows more structural variations in the choice of record types at the document level, and therefore the grammar extracted from the unsupervised alignments is noisier. On WinHelp, Angeli performs poorly, probably due to lack of domain-specific calibration; see the error analysis discussion in Section 4.5.2. Finally, we notice a slight anomaly in METEOR on WeatherGov; contrary to the trend present in the rest results, it does not correlate with BLEU when comparing DP-Auto to Angeli. This is probably because unlike BLEU, METEOR takes into account unigram recall, hence Angeli might be recovering more words present in the gold standard text compared to our model. This results in scoring higher in METEOR but lower in BLEU.

The results of our human evaluation study are shown in Table 5.4. Similarly to the previous experiment described in Section 4.5.2, we carried out an Analysis of Variance (Anova) to examine the effect of system type, (DP-Unsup, *k*-best-lm-dmv, Angeli, and Human) on fluency, semantic correctness and coherence ratings. Mean differences of 0.2 or more are significant at the 0.05 level using a post-hoc Tukey test. Interestingly, we observe that document planning improves system output overall, not only in terms of coherence. Across all dimensions DP-Unsup is perceived better than

	WeatherGov			WinHelp		
Model	F	SC	C	F	SC	C
k-BEST-LM-DMV	$3.66^{\circ\dagger\diamond}$	$3.34^{\circ\dagger\diamond}$	$3.56^{\circ\dagger\diamond}$	$3.27^{\dagger}$	$2.97^{\dagger}$	$2.93^{\dagger\diamond}$
DP-Unsup	4.09^{*}	3.62^{*}	$4.00^{*\circ}$	$3.46^{\dagger}$	$3.01^{\circ\dagger}$	$3.34^{*\circ\dagger}$
Angeli	3.93^{*}	3.58^{*}	$3.80^{*\diamond\dagger}$	$3.44^{\dagger}$	$2.79^{\dagger\diamond}$	$2.97^{*\dagger\diamond}$
Human	4.08^{*}	3.65^{*}	$4.02^{*\circ}$	$4.20^{*\circ\diamond}$	$4.03^{*\circ\diamond}$	$4.00^{*\circ\diamond}$

Table 5.4: Mean ratings for fluency (F), semantic correctness (SC) and coherence (C) on system output elicited by humans. (*: significantly different from *k*-BEST-LM-DMV; $^\circ$: significantly different from Angeli; $^\diamond$: significantly different from DP-Unsup; †: significantly different from Human).

k-BEST-LM-DMVand Angeli. As far as coherence is concerned, DP-Unsup performs best overall, on both domains and the differences in means between the comparison systems (Angeli and *k*-BEST-LM-DMV) are significant.

5.5.1 System Output

Figures 5.4-5.5 illustrate examples of system output along with the gold standard content selection for reference, for the WeatherGov and WinHelp domain, respectively. In general, both *k*-BEST-LM-DMV and DP-Unsup closely resemble the human output, in terms of fluency and coherence. They might differ in the verbalisation of the input (e.g., *'Showers before noon'*, versus *'A chance of showers'*, but without causing significant harm to the final output. However, DP-Unsup makes better decisions at the content level.

For example, on WeatherGov in Figure 5.4, notice that *k*-BEST-LM-DMV does not select the record **Precipitation Potential**, hence there is no mention in the text, unlike DP-Unsup (and Angeli) which render it as *'Chance of precipitation is 50%'*. Closer error analysis between the two models revealed a particularly strong trend in chosing different sequences of records. In particular we examined several scenarios from the test set that mention in the human text chance of rain or thunderstorms. These usually correlate with high integer values on the fields of **Precipitation Potential** record, and values such as `Chc` or `Def` on the *mode* fields of **Rain Chance** and **Thunderstorm Chance** records.

Even without much expert domain knowledge one realises that if these two records are salient in the database, they should be selected by the model and get mentioned in

Input:

Temperature			
Time	**Min**	**Mean**	**Max**
06-21	32	35	38

Cloud Sky Cover	
Time	**Percent (%)**
06-21	75-100

Rain Chance	
Time	**Mode**
06-21	Chc

Wind Speed			
Time	**Min**	**Mean**	**Max**
06-21	3	5	8

Wind Direction	
Time	**Mode**
06-21	SW

Precipitation Potential			
Time	**Min**	**Mean**	**Max**
06-21	31	43	52

DP-Unsup: Showers before noon. Cloudy, with a high near 38. Southwest wind between 3 and 8 mph. Chance of precipitation is 50 %.

k-best-lm-dmv: A chance of showers. Otherwise, cloudy, with a high near 38. Southwest wind between 3 and 8 mph.

Angeli: A chance of rain or drizzle after 9am. Mostly cloudy, with a high near 38. Southwest wind between 3 and 8 mph. Chance of precipitation is 50 %

Human: A 50 percent chance of showers. Cloudy, with a high near 38. Southwest wind between 3 and 6 mph.

Figure 5.4: Example system output and gold standard content selection on WeatherGov.

the text. In the case of *k*-best-lm-dmv we found that most of the times it correctly selected the first record of the document (usually **Rain Chance** or **Thunderstorm Chance**), but often missed the **Precipitation Potential** record in the middle or in the end of the document. This inconsistency is probably due to the locality of the HMM; the model has great confidence of selecting a rain-related event in the beginning of the document, but inevitably loses track of this mention further ahead in the record Markov chain. Examination of the derivation trees for the document planning rules extracted by DP-Unsup, reveals a considerable consensus of the *document plan* tree shown in Figure 5.6a. Notice how the long range dependency between the first **Rain Chance** record and the final **Precipitation Potential** record is preserved, regardless of the number of records between them.

A similar picture emerges for the WinHelp domain, (see an example of system output in Figure 5.5). As we have previously mentioned, the order of selecting records in this domain is crucial for the coherence of the final output. Notice how *k*-best-lm-dmv omits completely the **navigate-window-target** record (it should be

Input:

(1) navigate-desktop

envCmd	objName	type
right click	my network places	item

(2) navigate-desktop-target

envCmd	objName	type
left click	properties	menu

(3) navigate-window

envCmd	objName	type
right click	local area connection	item

(4) navigate-window-target

envCmd	objName	type
left click	properties	menu

(5) action-contextMenu

envCmd	objName	type	typeInto
left click	file and printer sharing for microsoft networks	checkbox	–

(6) exit-contextMenu

envCmd	objName	type
left click	ok	button

DP-UNSUP: Right-click my network places, and then click properties. Right-click local area connection, and click properties. Click to select the file and printer sharing for Microsoft networks, and then click ok.

k-BEST-LM-DMV: Right-click my network places, click properties. Right-click local area connection. Click to select the file and printer sharing for Microsoft networks, and then click ok.

ANGELI: Right-click my network places, and then click properties on the tools menu, and then click properties. Right-click local area connection, and then click properties. Click file and printer sharing for Microsoft networks, and then click ok.

HUMAN: Right-click my network places, and then click properties. Right-click local area connection, and then click properties. Click to select the file and printer sharing for Microsoft networks check box. Click ok.

Figure 5.5: Example system output and gold standard content selection on WINHELP. Numbers in the records indicate the correct order they should be selected by the document plan and then mentioned in the text.

mentioned after the phrase *'Right-click local area connection'*), or ANGELI emits twice the **navigate-desktop-target** record with some unexplained lexicalisation noise (*'and then click properties on the tools menu, and then click properties'*; there is no semantic evidence in the database for the word 'tools'). In contrast, DP-UNSUP mentions all records in the correct order, thus retaining the meaning of the document intact.

We performed an error analysis on this domain as well, but no clear conclusions can be drawn mostly due to the small size of the dataset. One possible explanation for why k-BEST-LM-DMV randomly omits records, is due to sparsity. Some records are infrequent (e.g., the **navigate-program** record appears only seven times in the corpus), therefore the model does not have a well-informed distribution for the emission of those. Of course, DP-UNSUP faces the same problem, since the weights of the extracted rules of G_{DP} are also inferred from the corpus in a frequentist approach[2]. However, by looking at the derivation trees produced by the Viterbi search algorithm, there is some evidence of learning document plans containing plausible sequences such as those shown in Figure 5.6b. The first tree possibly captures the regular pattern of navigating on the desktop in order to find an icon, then clicking on it and then performing some navigation and action on the window that popped up. In the second tree we observe a pattern of navigating first on the start menu and then selecting an item. In particular, notice the chaining of two **navigate-contextMenu** records before the corresponding **-target** record; this is outwith the expressive power of an HMM.

5.6 Discussion

In summary, we observe that integrating document planning via G_{DP} boosts performance. The extended models are consistently better than k-BEST-LM-DMV both in terms of automatic and human evaluation and are close or better than the supervised model of Angeli et al. (2010). The trained grammar produces document plans in the form of derivation trees that capture longer dependencies between records, which cannot be represented by a simple Markov chain, as in the model of Chapter 4. G_{DP} unavoidably introduces a lot of expressivity into the model in terms of exponentially many potential rules. We presented a simple way to overcome this problem, by obtaining the most frequent rules from the dataset, provided we have access to alignments

[2] We estimate rule weights based on evidence found in the data, by merely counting frequencies of rules and then normalising the counts (frequentist approach). An alternative would be to adopt a Bayesian approach in estimating the grammar rules (e.g., using a Variational method (Johnson, 2007)). However, we leave this to future work.

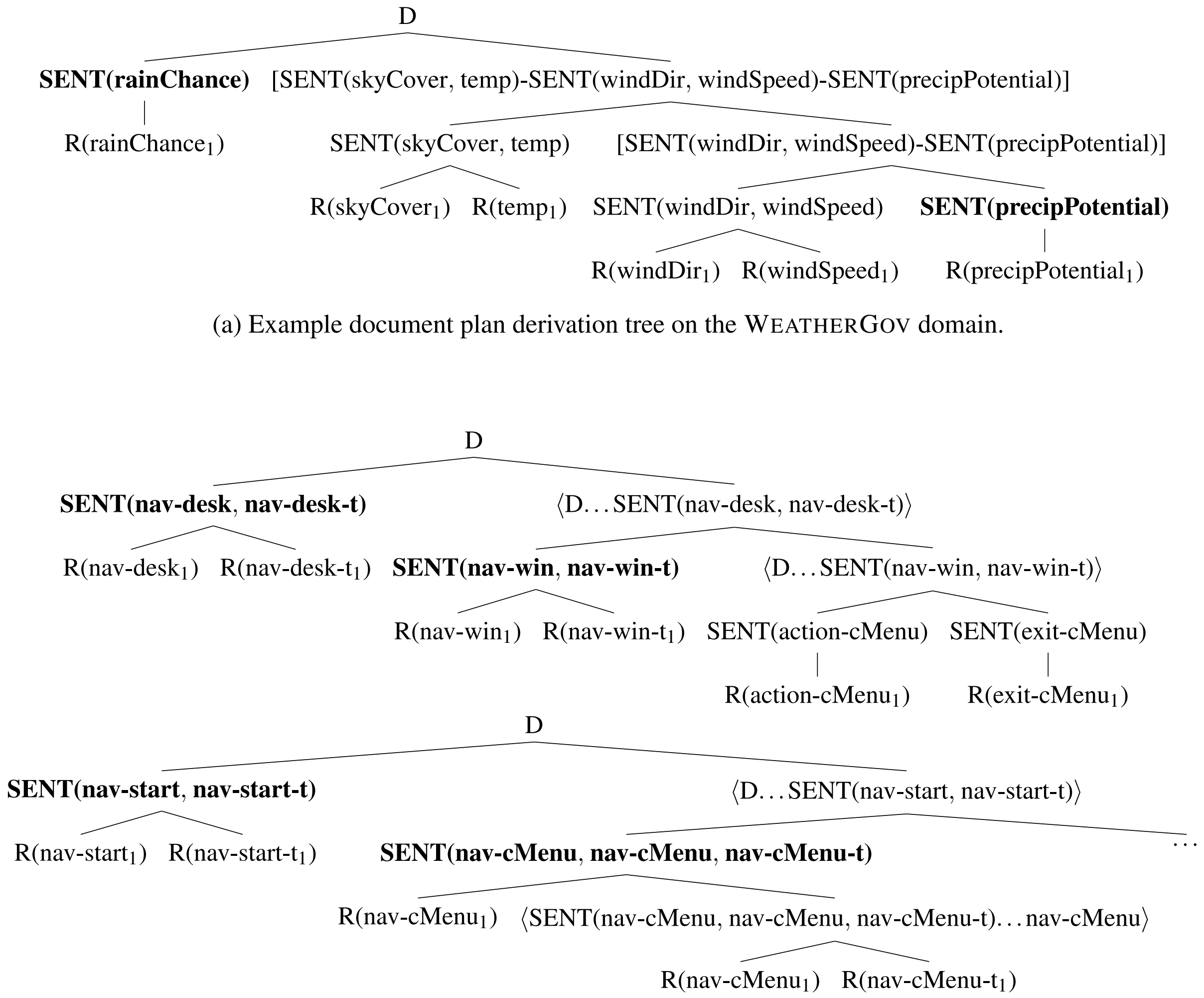
between the database and text. We can generate alignments in an unsupervised manner as shown in Section 3.3.1. We also showed that feeding the system with a grammar of better quality (via more refined alignments) can achieve state-of-the-art performance, without further changes to the model.

G_{DP} grammar is not the only way to define document plans of database records. An alternative would be to automatically induce a grammar using an existing grammar induction model such as the generative constituent-context model (CCM) of Klein and Manning (2002). Given a sequence of records for each scenario as an input string (Figure 5.7a) (obtained as explained in Section 5.3), we can induce an (unlabelled) bracketed tree structure, like the one shown in Figure 5.7b. Then we can assign a label to the non-terminals based on the yield they span (Figure 5.7c), and obtain a grammar of records similar in spirit to G_{DP}. The fundamental difference is that in this case we do not respect sentence delimiters (i.e., the full-stop), since it is not modelled explicitly as in the case of rule (2) in G_{DP}[3]. A better alternative would be to modify the original model of CCM to explicitly take into account sentence punctuation; we leave this approach to future work.

5.7 Summary

In this chapter we presented an extension to the model of Chapter 4 which integrates document planning. We presented an alternative to the original grammar G_{GEN} which identifies sentences in a document and represents the relationships between records within as well as among, sentences. We provided a simple mechanism to extract grammar rules directly from training data, and a modified scheme to train our generator. Finally, we evaluated our system on two multi-sentence datasets, namely WEATHERGOV and WINHELP, obtaining state-of-the-art performance compared to the system of Angeli et al. (2010). Error analysis of the document plans created during decoding revealed interesting learnt domain-specific patterns at the record level, which were out of scope of the simpler document planning capabilities of the original model presented in Chapter 4.

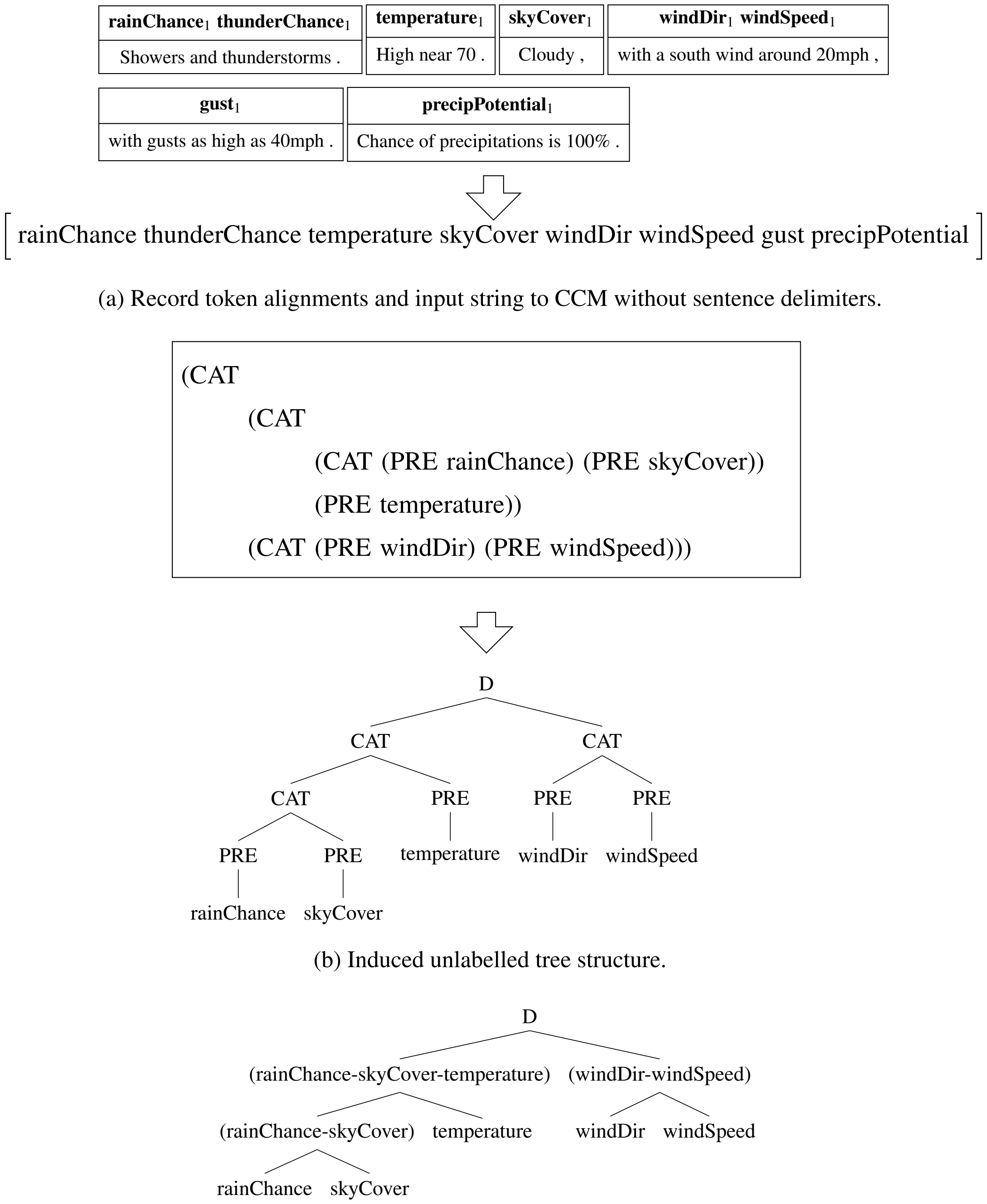
[3] Including the sentence delimiter in the input string, is likely to lead the model to overfitting. This has been also noted by the authors of CCM who remove punctuation from their training corpus (sentences with 10 words or less from the WSJ corpus).


D
SENT(rainChance)
[SENT(skyCover, temp)-SENT(windDir, windSpeed)-SENT(precipPotential)]
R(rainChance$_1$)
SENT(skyCover, temp)
[SENT(windDir, windSpeed)-SENT(precipPotential)]
R(skyCover$_1$)
R(temp$_1$)
SENT(windDir, windSpeed)
SENT(precipPotential)
R(windDir$_1$)
R(windSpeed$_1$)
R(precipPotential$_1$)
D
SENT(nav-desk, **nav-desk-t)**
⟨D...SENT(nav-desk, nav-desk-t)⟩
R(nav-desk$_1$)
R(nav-desk-t$_1$)
SENT(nav-win, **nav-win-t)**
⟨D...SENT(nav-win, nav-win-t)⟩
R(nav-win$_1$)
R(nav-win-t$_1$)
SENT(action-cMenu)
SENT(exit-cMenu)
R(action-cMenu$_1$)
R(exit-cMenu$_1$)
D
SENT(nav-start, **nav-start-t)**
⟨D...SENT(nav-start, nav-start-t)⟩
R(nav-start$_1$)
R(nav-start-t$_1$)
SENT(nav-cMenu, **nav-cMenu**, **nav-cMenu-t)**
...
R(nav-cMenu$_1$)
⟨SENT(nav-cMenu, nav-cMenu, nav-cMenu-t)...nav-cMenu⟩
R(nav-cMenu$_1$)
R(nav-cMenu-t$_1$)

(a) Example document plan derivation tree on the WEATHERGOV domain.

(b) Example document plan derivation trees for the WINHELP domain. We abbreviated record type names for the sake of clarity.

Figure 5.6: Example derivation trees that correspond to the document planning rules of G_{DP} captured during the Viterbi search of the examples shown in Figure 5.4 and 5.5, for the DP-UNSUP system. Interesting learned patterns are highlighted in bold.


rainChance₁ thunderChance₁
Showers and thunderstorms .
temperature₁
High near 70 .
skyCover₁
Cloudy ,
windDir₁ windSpeed₁
with a south wind around 20mph ,
gust₁
with gusts as high as 40mph .
precipPotential₁
Chance of precipitations is 100% .
rainChance thunderChance temperature skyCover windDir windSpeed gust precipPotential
(CAT
(CAT
(CAT (PRE rainChance) (PRE skyCover))
(PRE temperature))
(CAT (PRE windDir) (PRE windSpeed)))
D
CAT
CAT
CAT
PRE
PRE
PRE
PRE
PRE
temperature
windDir
windSpeed
rainChance
skyCover
D
(rainChance-skyCover-temperature)
(windDir-windSpeed)
(rainChance-skyCover)
temperature
windDir
windSpeed
rainChance
skyCover

(a) Record token alignments and input string to CCM without sentence delimiters.

(b) Induced unlabelled tree structure.

(c) Automatically labelled tree structure.

Figure 5.7: Grammar extraction example from the WINHELP domain using CCM: (a) We first take the alignments of records on words, map them to their corresponding types and ignore the sentence segmentation; this is the input string used for training. (b) Then CCM generates a bracketing of the input string, which represents an unlabelled tree. (c) We label non-terminals using the surface level yield of their span.

Chapter 6

An Exploration in Discriminative Reranking

In the previous chapter we extended the original grammar presented in Chapter 4 by introducing rules that perform document planning. In this chapter we will go back to the original model, and will focus on a different learning scheme, i.e., reranking the hypergraph implementation discriminatively. Since we encode a *k*-best list of derivations at each hypernode, it is quite appealing to try and re-order them using evidence from the input database and text in the form of features. Given the extra flexibility of introducing arbitrary features to the model, we will look into an area which could not be explored with either of the previous two models presented, namely performing content selection on the field level. We used the structured perceptron (Collins, 2002) for learning, and experimented on the ATIS dataset. Since the perceptron is a discriminative training algorithm, experiments on larger datasets (in terms of output text length) would require major modifications (i.e., introduce parallelisation) to the original algorithm and we thus leave this to future work[1].

6.1 Motivation

Following the generative approach presented in Chapter 4, we first have to learn the weights of the PCFG by maximising the joint likelihood of the model and then perform generation by finding the best derivation tree in the hypergraph. If we regard this model as a baseline system instead, we could potentially further improve it using

[1] We refrained from experimenting on ROBOCUP, because content selection is fixed on the baseline models we compare against as we saw in Section 4.4.4. As a result, there would be no gain from the exploration on the additional field-level features.

discriminative reranking (Collins, 2000). Typically, this method first creates a list of *n*-best candidates from a generative model, and then reranks them with arbitrary features (both local and global) that are either not computable or intractable to compute within the baseline system.

An appealing alternative is to rerank the hypergraph *directly* (Huang, 2008). As it compactly encodes exponentially many derivations, we can explore a much larger hypothesis space than would have been possible with an *n*-best list. Importantly, in this framework non-local features are computed at all internal hypergraph nodes, allowing the decoder to take advantage of them continuously at all stages of the generation process. We incorporate features that are local with respect to a span of a sub-derivation in the packed forest; we also (approximately) include features that arbitrarily exceed span boundaries, thus capturing more global knowledge.

Discriminative reranking has been employed in many NLP tasks such as syntactic parsing (Charniak and Johnson, 2005; Huang, 2008), machine translation (Shen et al., 2004; Li and Khudanpur, 2009) and semantic parsing (Ge and Mooney, 2006). The presented approach is closest to Huang (2008) who also performs forest reranking on a hypergraph, using both local and non-local features, whose weights are tuned with the averaged perceptron algorithm (Collins, 2002). We adapt forest reranking to generation and introduce several task-specific features that boost performance. Compared to the discriminative model of Angeli et al. (2010), our model is fundamentally different in the learning aspect. We have a single reranking component that applies throughout, whereas they train different discriminative models for each local decision.

We will experiment on the ATIS domain only as a proof of concept, since it is much smaller in terms of output text length compared to WEATHERGOV and WINHELP. Parallelising the perceptron algorithm in order to scale to larger text outputs requires major engineering and tuning, and we leave it to future work. However, ATIS poses the greatest challenges compared to the rest in terms of content selection at the field level; many of its record types have a large number of fields (e.g., record type **Flight** has 13 fields, whereas the maximum number of fields in WEATHERGOV's record types is 4). Therefore we expect that this domain will benefit a lot more than the others from features at the field level. In addition all the field types in ATIS are categorical, i.e., we keep a separate multinomial distribution for each value in order to lexicalise them (see the rule weight of rule (9) in grammar G_{GEN} in Table 4.1). Again ATIS has fields with a much larger number of values (e.g., field *from* of the record type **Flight** has 61 values, whereas the field *mode* of the record type **Wind Direction** with most values in WEATHERGOV has merely 17 values). Given that most of these values are rarely seen we believe that training the corresponding features discriminatively, will result in a greater boost in performance than in the other datasets.

6.2 Hypergraph Reranking

Recall that for our generation task, we are given a set of database records **d**, and our goal is to find the best corresponding text g. For this study, we will use the grammar G_{GEN} in Table 4.1[2], which we represent using the hypergraph framework described in Section 4.3.3. The fundamental difference with the previous models is that the weights on the hyperarcs are defined by a variety of feature functions, which we learn via a discriminative online update algorithm. During decoding we find the best scoring derivation $(\hat{g}, \hat{h})$ by maximizing over configurations of h:

$$(\hat{g}, \hat{h}) = \arg\max_{g,h} \alpha \cdot \Phi(\mathbf{d}, g, h) \tag{6.1}$$

We define the score of (g,h) as the dot product between a high dimensional feature representation $\Phi = (\Phi_1, \ldots, \Phi_m)$ and a weight vector α.

We estimate the weights α using the averaged structured perceptron algorithm (Collins, 2002), which is well known for its good performance in similar large-parameter NLP tasks (Liang et al., 2006; Huang, 2008). As shown in Algorithm 6.1, the perceptron makes several passes over the training scenarios, and in each iteration it computes the best scoring $(\hat{g}, \hat{h})$ among the candidate derivations, given the current weights α. In line 7, the algorithm updates α with the difference (if any) between the feature representations of the best scoring derivation $(\hat{g}, \hat{h})$ and the the *oracle derivation* (w, h^+). Recall that, $\hat{g}$ is the estimated text and w the gold-standard text. We also define $\hat{h}$ as the estimated latent configuration of the model and h^+ as the oracle latent configuration. The final weight vector α is the average of weight vectors over T iterations and N scenarios. This averaging procedure avoids overfitting and produces more stable results (Collins, 2002).

In the following, we first explain how we decode in this framework, i.e., find the best scoring derivation (Section 6.2.1) and then discuss our definition for the oracle derivation (w, h^+) (Section 6.2.2). Our features are described in Section 6.2.3.

[2] Since we experiment only on a single-sentence output text, we do not need a document plan, hence we will not make use of the sophisticated grammar G_{GEN}++ in Table 5.1

```
**function** PERCEPTRON(Training scenarios: $(\mathbf{d_i}, w, h_i^+)_{i=1}^N$)
    $\alpha \leftarrow 0$
    **for** $t \leftarrow 1 \ldots T$ **do**
        **for** $i \leftarrow 1 \ldots N$ **do**
            $(\hat{g}, \hat{h}) = \arg\max_{g,h} \alpha \cdot \Phi(\mathbf{d_i}, g_i, h_i)$
            **if** $(w_i, h_i^+) \neq (\hat{g}_i, \hat{h}_i)$ **then**
                $\alpha \leftarrow \alpha + \Phi(\mathbf{d_i}, w_i, h_i^+) - \Phi(\mathbf{d_i}, \hat{g}_i, \hat{h}_i)$
            **end if**
        **end for**
    **end for**
    **return** $\frac{1}{T}\sum_{t=1}^{T} \frac{1}{N}\sum_{i=1}^{N} \alpha_t^i$
**end function**
```

Figure 6.1: The average structured perceptron algorithm (Collins, 2002).

6.2.1 Hypergraph Decoding

Following Huang (2008), we also distinguish features into local, i.e., those that can be computed within the confines of a single hyperedge, and non-local, i.e., those that require the prior visit of nodes other than their antecedents. For example, the *Alignment* feature in Figure 6.2(a) is local, and thus can be computed a priori, but the *Word Trigrams* is not; in Figure 6.2(b) words in parentheses are sub-generations created so far at each word node; their combination gives rise to the trigrams serving as input to the feature. However, this combination may not take place at their immediate ancestors, since these may not be adjacent nodes in the hypergraph. According to the grammar in Table 4.1, there is no direct hyperedge between nodes representing words (W) and nodes representing the set of fields these correspond to (FS); rather, W and FS are connected implicitly via individual fields (F). Note that, in order to estimate the trigram feature at the FS node, we need to carry word information in the derivations of its antecedents, as we go bottom-up.[3]

Given these two types of features, we can then adapt Huang's 2008 approximate decoding algorithm to find $(\hat{g}, \hat{h})$. Note that this algorithm is similar in spirit to the implementation of the cube pruning algorithm on hypergraphs presented in detail in Section 4.15; the only essential difference is the incorporation of feature vectors. Essentially, we perform bottom-up Viterbi search, visiting the nodes in reverse topolog-

[3] We also store field information to compute content selection features, described in Section 6.2.3.

ical order, and keeping the k-best derivations for each. The score of each derivation is a linear combination of local and non-local feature weights. In machine translation, a decoder that implements forest rescoring (Huang and Chiang, 2007) uses the language model as an external criterion of the goodness of sub-translations on account of their grammaticality. Analogously here, non-local features influence the selection of the best combinations, by introducing knowledge that exceeds the confines of the node under consideration and thus depend on the sub-derivations generated so far. (e.g., word trigrams spanning a field node rely on evidence from antecedent nodes that may be arbitrarily deeper than the field's immediate children).

6.2.2 Oracle Derivation

So far we have remained agnostic with respect to the oracle derivation (w, h^+). In other NLP tasks such as syntactic parsing, there is a gold-standard parse, that can be used as the oracle. In our generation setting, such information is not available. We do not have the gold-standard alignment between the database records and the text that verbalizes them. Instead, we approximate it using the existing decoder to find the best latent configuration h^+ given the observed words in the training text w.[4] This is similar in spirit to the alignment task presented in Section 3.3.1.

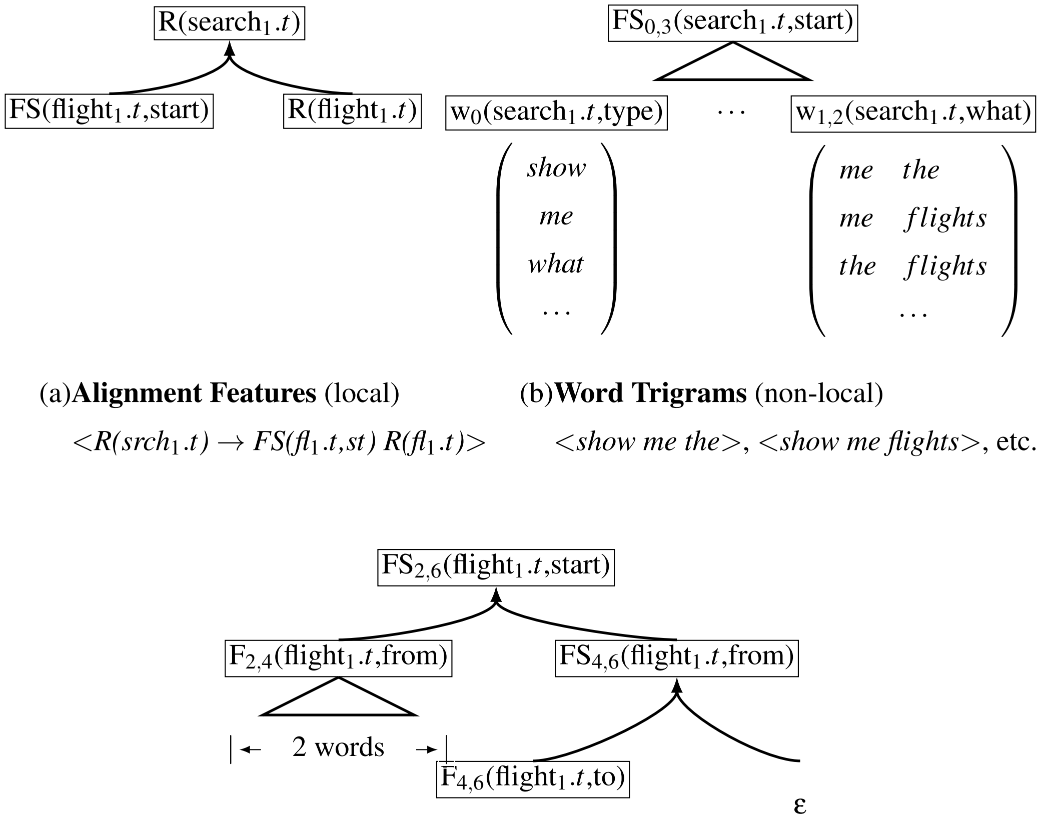
6.2.3 Features

Broadly speaking, we defined two types of features, namely lexical and content selection ones. In addition, we used a generatively trained PCFG as a baseline feature and an alignment feature based on the co-occurrence of records (or fields) with words.

Baseline Feature This is the log score of k-BEST-LM trained on grammar G_{GEN} of Table 4.1 as described in Chapter 4. Intuitively, the feature refers to the overall goodness of a specific derivation, applied locally in every hyperedge.

Alignment Features Instances of this feature family refer to the count of each PCFG rule from Table 4.1. For example, how many times we are going to include in a derivation rule $\mathrm{R}(search_1.t) \rightarrow \mathrm{FS}(flight_1, start)\ \mathrm{R}(flight_1.t)$ (see Figure 6.2(a)).

[4] In machine translation, Huang (2008) provides a soft algorithm that finds the forest oracle, i.e., the parse among the reranked candidates with the highest Parseval F-score. However, it still relies on the gold-standard reference translation.


R(search$_1$.t)
FS(flight$_1$.t,start)
R(flight$_1$.t)
FS$_{0,3}$(search$_1$.t,start)
w$_0$(search$_1$.t,type)
...
w$_{1,2}$(search$_1$.t,what)
show
me
what
...
me the
me flights
the flights
...
FS$_{2,6}$(flight$_1$.t,start)
F$_{2,4}$(flight$_1$.t,from)
FS$_{4,6}$(flight$_1$.t,from)
2 words
F$_{4,6}$(flight$_1$.t,to)
ε

(a)**Alignment Features** (local)

<R(srch$_1$.t) → FS(fl$_1$.t,st) R(fl$_1$.t)>

(b)**Word Trigrams** (non-local)

<show me the>, *<show me flights>*, etc.

(c)**Field Bigrams** (non-local)

<from to> | flight

(d)**Number of Words per Field** (local)

<2 | from>

Figure 6.2: Simplified hypergraph examples with corresponding local and non-local features.

Lexical Features These features encourage grammatical coherence and inform lexical selection over and above the limited horizon of the language model captured by Rules (6)–(10). They also tackle anomalies in the generated output, due to the chaining of the CFG rules at the record and field level.

Word Bigrams/Trigrams: this is a group of non-local feature functions that count word *n*-grams at every level in the hypergraph (see Figure 6.2(b)). The integration of words in the sub-derivations is adapted from Chiang (2007).

Number of Words per Field: this feature function counts the number of words for every field, aiming to capture compound proper nouns and multi-word expressions, e.g., fields *from* and *to* frequently correspond to two or three words such as '*new york*' and '*salt lake city*' (see Figure 6.2(d)).

Consecutive Word/Bigram/Trigram: this feature family targets adjacent repetitions of the same word, bigram or trigram, e.g., '*show me the show me the flights*'.

Content Selection Features at Field Level Features in this category target primarily content selection and influence appropriate choice at the field level.

Field bigrams/trigrams: analogously to the lexical features mentioned above, we introduce a series of non-local features that capture field *n*-grams, given a specific record. For example the record **flight** in the air travel domain typically has the field names <*from to*> (see Figure 6.2(c)). The integration of fields in sub-derivations is implemented in a fashion similar to the integration of words.

Number of Fields per Record: this feature family is a coarser version of the *Field bigrams/trigrams* feature, which is deemed to be sparse for rarely-seen records.

Field with No Value: although records in the ATIS database schema have many fields, only a few are assigned a value in any given scenario. For example, the **flight** record has 13 fields, of which only 1.7 (on average) have a value. Practically, in a generative model this kind of sparsity would result in very low field recall. We thus include an identity feature function that explicitly counts whether a particular field has a value.

6.3 Experiments

In this exploratory study, our aim was to assess the merits of discriminative reranking as a framework for concept-to-text generation. In addition we wanted to experiment with content selection features at the field level. As we discussed Section 6.1, we believe that some database record types with a rich set of fields and many different categorical values in each field as is the case in ATIS, can benefit a lot by explicit feature engineering specifically at the field level. In the experiments we describe below we restrict our models in two ways: firstly, we include only shallow surface lexical features rather than syntactic dependencies; in this way the model can be loosely compared to k-BEST-LM. This is because we wanted to exclude any longer range influences that stem from syntax which might interfere with content selection at the field level. Secondly, as mentioned earlier, we evaluate our model only on ATIS. This was mandated by the discriminative nature of the learning algorithm we used, namely the structured perceptron. Recall that each update of the feature weights Φ require two Viterbi updates, i.e., one for the computation of the non-local features (local features can be pre-computed and cached), and another for the oracle derivation. Given that this update takes place upon observing each training scenario, it cannot be implemented in a parallel architecture[5], and therefore does not scale for domains with longer output texts, such as WEATHERGOV and WINHELP. We did not experiment on ROBOCUP as content selection is fixed on the baseline models we compare to (see Section 4.4.4), therefore there would be no actual gain from field-level features.

6.3.1 System Comparison

We evaluated three configurations of our model: A system that only uses the top scoring derivation in each sub-generation and incorporates only the baseline and alignment features (1-BEST+BASE+ALIGN). Our second system considers the k-best derivations and additionally includes lexical features (k-BEST+BASE+ALIGN+LEX). The number of k-best derivations was set to 40 and estimated experimentally on held-out data. And finally, our third system includes the full feature set (k-BEST+BASE+ALIGN+LEX+FIELD). Note that the second and third system incorporate non-local features, hence the use of k-best derivation lists.[6] We compared our model to Angeli et al. (2010) whose approach is closest to ours.

We evaluated system output automatically, using the BLEU-4 and the METEOR

[5] McDonald et al. (2010) present an implementation of the perceptron algorithm that runs in parallel or rather batch-processing environments, by splitting the training scenarios in shards and processing each in isolation. They show that combining and averaging the resulting weights from each shard does not compromise the accuracy of the algorithm. We leave the implementation of this framework to future work.

[6] Since the addition of these features essentially entails reranking, it follows that the systems would exhibit the exact same performance as the baseline system with 1-best lists.

score. In addition, we evaluated the generated text by eliciting human judgements. We randomly selected 12 documents from the test set and generated output with two of our models (1-BEST+BASE+ALIGN and *k*-BEST+BASE+ALIGN+LEX+FIELD) and Angeli et al.'s (2010) model. We also included the original text HUMAN as gold standard.

6.4 Results

System	BLEU	METEOR
1-BEST+BASE+ALIGN	$21.93^{\circ\dagger\diamond}$	$34.01^{\circ\dagger\diamond}$
k-BEST+BASE+ALIGN+LEX	$28.66^{*\circ\diamond}$	$45.18^{*\circ\diamond}$
k-BEST+BASE+ALIGN+LEX+FIELD	$30.62^{*\circ\dagger}$	$46.07^{*\circ\dagger}$
ANGELI	$26.77^{*\dagger\diamond}$	$42.41^{*\dagger\diamond}$

Table 6.1: BLEU-4 and METEOR results on ATIS. (*: significantly different from 1-BEST+BASE+ALIGN; $^\circ$: significantly different from ANGELI; †: significantly different from *k*-BEST+BASE+ALIGN+LEX; $^\diamond$: significantly different from *k*-BEST+BASE+ALIGN+LEX+FIELD).

System	F	SC
1-BEST+BASE+ALIGN	$2.70^{\diamond\dagger\circ}$	$3.05^{\diamond\dagger}$
k-BEST+BASE+ALIGN+LEX+FIELD	$4.02^{*\circ}$	$4.04^{*\circ}$
ANGELI	$3.74^{*\diamond\dagger}$	$3.17^{\diamond\dagger}$
HUMAN	$4.18^{*\circ}$	$4.02^{*\circ}$

Table 6.2: Mean ratings for fluency (F) and semantic correctness (SC) on system output elicited by humans on ATIS.(*: significantly different from 1-BEST+BASE+ALIGN; $^\circ$: significantly different from ANGELI; $^\diamond$: significantly different from *k*-BEST+BASE+ALIGN+LEX+FIELD; †: significantly different from HUMAN).

Table 6.1 summarizes our results. As can be seen, inclusion of lexical features gives our decoder an absolute increase of 6.73% in BLEU over the 1-BEST system. It is interesting to note though that this baseline is much stronger than the baseline of the original generative model presented in Chapter 4 (BLEU-4 21.93 versus 11.85). This is expected given that the new model benefits more from the discriminative training even though it uses the same amount of data. Our model also significantly outperforms ANGELI. Our lexical features seem more robust compared to their templates. This is especially the case with infrequent records, where their system struggles to learn any meaningful information. Addition of the content selection features further boosts performance. Our model increases by 8.69% over the 1-BEST system and 3.85% over ANGELI in terms of BLEU (all differences are statistically significant). We observe a similar trend when evaluating system output with METEOR. Differences in magnitude are larger with the latter metric.

The results of our human evaluation study are shown in Table 6.2. We carried out an Analysis of Variance (ANOVA) to examine the effect of system type (1-BEST, *k*-BEST, ANGELI, and HUMAN) on the fluency and semantic correctness ratings. Mean differences were compared using a post-hoc Tukey test. The *k*-BEST system is significantly better than the 1-BEST and ANGELI ($a < 0.01$) both in terms of fluency and semantic correctness. ANGELI is significantly better than 1-BEST with regard to fluency ($a < 0.05$) but not semantic correctness. There is no statistically significant difference between the *k*-BEST output and the original sentences (HUMAN). Examples of system output are shown in Figure 6.3. They broadly convey similar meaning with the gold-standard; ANGELI in Figure 6.3a exhibits some long-range repetition, probably due to re-iteration of the same record patterns. We tackle this issue with the inclusion of non-local content selection features, justifying the validity of our claim that features at the field level can contribute to better decisions, and thus produce more fluent output. Other than that, the 1-BEST system has some grammaticality issues, which we avoid by defining features over lexical *n*-grams and repeated words. It is worth noting that both our system and ANGELI produce output that is semantically compatible with but lexically different from the gold standard (compare *please list the flights* and *show me the flights* against *give me the flights* in Figure 6.3a); this is of course expected given the size of the vocabulary (927 words). Finally, notice how the multi-word destination name, *Salt Lake City*, in Figure 6.3b is captured only in part in ANGELI and 1-BEST. This is probably due to insufficient training data instances for the particular field value. K-BEST tackles this with the aid of lexical features.

6.5 Summary

In this chapter we presented an alternative learning approach by employing discriminative reranking of the hypergraph implementation of the generator presented in Chapter 4. The key idea was to re-order the *k*-best derivations encoded in each hyperarc, based on knowledge obtained from the input database and the text. We achieved that by defining local and global features that capture content selection and surface realisation, as well as features that specifically perform content selection on the field level. We employed the structured perceptron algorithm (Collins, 2002) for learning and evaluated our generator on the ATIS dataset. Our system outperformed a strong baseline using alignment features and the score of the model presented in Chapter 4, as well as the discriminative model of Angeli et al. (2010). Error analysis on system output indicated that the incorporation of non-local features at the field level contributed to resolving long-range issues, such as avoiding n-gram repetitions and correctly realising multi-word field values.

Input:

Flight

from	to
Phoenix	Milwaukee

Time

when	dep/ar
evening	departure

Day

day	dep/ar/ret
Wednesday	departure

Search

type	what
query	flight

1-BEST: On Wednesday evening from from Phoenix to Milwaukee on Wednesday evening

k-BEST: Please list the flights from Phoenix to Milwaukee on Wednesday evening

ANGELI: Show me the flights from Phoenix to Milwaukee on Wednesday evening flights from Phoenix to Milwaukee

HUMAN: Give me the flights from Phoenix to Milwaukee on Wednesday evening

(a)

Input:

Flight

from	to
Oakland	Salt Lake City

Day

day	dep/ar/ret
Wednesday	departure

Search

type	what	of
argmax	flight	departure time

1-BEST: From Oakland to Salt on Wednesday from from Oakland to Salt

k-BEST: Latest the last flight from Oakland to Salt Lake City on Wednesday

ANGELI: Show me the latest flight available flights from Oakland California to Salt on Wednesday

HUMAN: Get last flight from Oakland to Salt Lake City on Wednesday

(b)

Figure 6.3: Examples of scenario input and system output on ATIS.

Chapter 7

Conclusions

In this thesis, we focused on the task of concept-to-text generation, namely the process of automatically producing textual output from structured non-linguistic input. Traditionally, this task has been addressed in a modular pipeline approach, with each module tackling different aspects of the generation process (i.e., 'what to say' and 'how to say'), independently or with limited interaction. Most successful systems adhere to rule-based methods and rely partially or entirely on manually annotated data. We presented an end-to-end modelling approach that treats various generation components in a joint fashion, does not rely on annotation, and extracts the structural and linguistic knowledge necessary for the generation of the final output, from training data.

The key idea of our model is to recast generation as a probabilistic parsing problem. To achieve this, we developed a syntactic probabilistic context-free grammar (Chapter 4) that decomposes the structure of the database input as a sequence of records, fields and values and observes words of the collocated text as terminal symbols. The weights on the grammar rules are acquired in an unsupervised manner using EM and the inside-outside algorithm. We presented several parsing algorithms based on the popular CYK parser (Kasami, 1965; Younger, 1967), that take as input a database of records and a trained in-domain grammar and produce text. In order to guarantee the fluency and grammaticality of the output, we intersected our PCFG with an ensemble of external linguistically motivated models, namely an n-gram language model and the dependency model with valence of Klein and Manning (2004). We implemented our decoding algorithms using the hypergraph framework. We empirically evaluated our models both automatically and via eliciting human judgements across four domains, namely ROBOCUP, WEATHERGOV, ATIS and WINHELP. Our models performed comparably or achieved state-of-the-art performance compared to the discriminative model of Angeli et al. (2010). However, they do not perform as well compared to the system of Kim and Mooney (2010) on ROBOCUP, possibly due to the later being trained with more supervision and relying on an additional re-ordering step before emitting the output text.

In Chapter 5 we extended our PCFG with two syntactic rules that aim to capture patterns at the document level. The original model selected and ordered records from the database input in order to be mentioned in the text, on a local Markovian basis. In this incarnation of the model we defined a document plan as a sequence of sentences, where each sentence contains a sequence of records. Since the extra rules increased the search space for our decoder prohibitively, we directly extracted domain-specific rules from the training data as a pre-processing step and included this subset with the rest of the grammar for decoding the database input. We evaluated the new model automatically and via user studies on two multi-sentence domains, namely WEATHERGOV and WINHELP. We noticed a considerable increase in performance compared to the original models and the competitive baseline system of Angeli et al. (2010), and observed promising learned document plan trees based on a post-hoc error analysis.

Finally, Chapter 6 explores how to enhance the original model presented in Chapter 4 with more expressivity by using a discriminative reranking approach. We achieve this by training the original PCFG discriminatively, by directly reranking the hypergraph representation. We defined a set of lexical and content selection feature vectors on the arcs of the hypergraph, and learned their weights using the averaged structured perceptron algorithm (Collins, 2002). Experiments on the ATIS domain revealed a significant increase in terms of automatic scores and according to a human evaluation study, when compared to the original model presented in Chapter 4 and the discriminative baseline of Angeli et al. (2010).

Overall, the work presented in this thesis makes a strong case for data-driven concept-to-text generation. To the best of our knowledge this is the first systematic study of modelling jointly content planning, sentence planning and surface realisation in one unified framework. Addressing text generation in an end-to-end fashion as a common parsing problem, guided by a syntactic grammar trained on in-domain data, allows for compact and highly portable systems, benefiting from limited or no supervision. The viability and versatility of our models is supported by a rigorous evaluation methodology across four real-world domains. Outwith generation, we hope that some of the work described here might be of relevance to other applications such as summarisation or machine translation.

To conclude, the experimental results obtained in this thesis provide evidence that a purely data-driven probabilistic joint model for concept-to-text generation can generate fluent and understandable text with minimal manual intervention. It is only a matter of time and concentrated effort before we go from the prototypes presented here to commercially deployed systems.

7.1 Future Work

Future extensions are many and varied. An obvious extension concerns porting the framework to more challenging domains with richer vocabulary and longer texts. More specifically, an interesting line of research would be to focus on readily available user-generated input data such as Freebase (`www.freebase.com`) and Wikipedia articles (`www.wikipedia.com`), and generate text on domains such as biographies, music-related facts, movies or in general media descriptions. The benefits as well as challenges are wide-ranging: The size of input data is orders of magnitude larger, which calls for more elaborate content selection on an early stage, to avoid a blowup in search space. Hixon and Passonneau (2013) adopt a PageRank-style schema summarisation technique which allows them to traverse the whole input database and select tables that are close to each other based on an input query. This has the benefit of relying on the input data only, thus avoiding the cost of jointly optimising the database records, fields and values with the resulting text. A similar idea could be employed to filter out excess information before fitting it to our PCFG. Factual text such as biographies or descriptions of music albums, contain lots of references to entities, e.g., the parents of an author, his/her birthplace and so on. Generating fluent text requires some notion of more sophisticated sentence planning in the form of referring expressions generation, such as coreference. This way we will be able to avoid unnecessary repetitions of source identifiers, for example using the same proper name all the time. A straightforward extension to the existing model would be to directly identify entities as part of the grammar, similarly to how we tackle fields and values. This way we can employ an external coreference resolution model to constraint the search, similarly to how we used an n-gram language model and a dependency model to rescore k-best derivations during decoding. In general, moving to an open-ended input database definitely raises questions as to the ability of the existing decoders to scale satisfactorily, and will probably require better engineering of the existing framework.

On another issue, although we have adopted throughout this thesis the hypergraph framework as a means of representing our PCFG, it is not the only implementation we could employ. It would be interesting to apply different formalisms such as finite state transducers (de Gispert et al., 2010), or push-down automata (Iglesias et al., 2011) and measure the tradeoffs between speed and performance. The former implementation could probably be an attractive solution for the scaling concerns raised above in the case of larger domains. Finally, in terms of document planning it would be worthwile to experiment with more sophisticated planners either via better grammar refinement or more expressive grammar formalisms (Cohn et al., 2010).

Appendix A

Example Scenarios

In the following sections we include example scenarios for each dataset we used. Each scenario consists of the database records in a tab-separated, attribute-value pair format (following Liang et al. (2009)). Each row corresponds to a uniquely identifiably record (see the first attribute `.id`), followed by the record type (e.g. `.type:windDir`), and a set of attribute-value pairs separated with tabs that correspond to the type of the field (`#, @, $`, denote integer, categorical, and string types, respectively), the field name, and its value. For example the pair `@mode:S`, stands for the categorical field *mode* with value `S`.

Below the records follows the corresponding text, split in lines. ROBOCUP and ATIS have only single sentence texts, whereas WEATHERGOV and WINHELP have multi-sentence texts. In WEATHERGOV, lines correspond to phrases split at punctuation, while in WINHELP, lines correspond to sentences split at the full-stop. Finally, record alignments are appended at the end of each line of text. Recall that record alignments are obtained either manually or automatically based on domain-specific, human-created heuristics. Each row indicates a set of alignments per line of text, and is represented by a sequence of numbers, corresponding to record identifiers. For example:

South southwest wind around 10 mph.	`3 2`

means that this line of text is aligned to records with id `3` and `2`, respectively.

A.1 RoboCup Examples

[Example 1]

```
.id:0  .type:pass      @arg1:purple9  @arg2:purple2
.id:1  .type:kick      @arg1:purple2
.id:2  .type:badPass   @arg1:purple2  @arg2:pink8
.id:3  .type:turnover  @arg1:purple2  @arg2:pink8
```

Purple9 passes out to Purple2 | 0

[Example 2]

```
.id:0  .type:pass  @arg1:purple10  @arg2:purple8
.id:1  .type:kick  @arg1:purple8
.id:2  .type:pass  @arg1:purple8   @arg2:purple10
```

Purple8 immediately returns the ball to Purple10 | 2

[Example 3]

```
.id:0  .type:pass  @arg1:pink11  @arg2:pink9
.id:1  .type:kick  @arg1:pink9
.id:2  .type:pass  @arg1:pink9   @arg2:pink7
```

Purple8 passes to Purple10 who was well defended by Pink3 | 0

[Example 4]

```
.id:0  .type:pass         @arg1:purple3     @arg2:purple1
.id:1  .type:playmode     @arg1:free_kick_l
.id:2  .type:ballstopped
```

free kick from the purple team | 1

A.2 WEATHERGOV Examples

[Example 1]

```
.id:0   .type:temperature      @time:17-30  #min:57        #mean:62
                                                           #max:71
.id:1   .type:windChill        @time:17-30  #min:0         #mean:0
                                                           #max:0
.id:2   .type:windSpeed        @time:17-30  #min:7         #mean:8
                                            #max:10        @bucket:0-10
.id:3   .type:windDir          @time:17-30  @mode:S
.id:4   .type:gust             @time:17-30  #min:0         #mean:0
                                                           #max:0
.id:5   .type:skyCover         @time:17-30  @bucket:25-50
.id:6   .type:skyCover         @time:17-21  @bucket:25-50
.id:7   .type:skyCover         @time:17-26  @bucket:25-50
.id:8   .type:skyCover         @time:21-30  @bucket:25-50
.id:9   .type:skyCover         @time:26-30  @bucket:25-50
.id:10  .type:precipPotential  @time:17-30  #min:0         #mean:3
                                                           #max:10
.id:11  .type:thunderChc       @time:17-30  @mode:--
.id:12  .type:thunderChc       @time:17-21  @mode:--
.id:13  .type:thunderChc       @time:17-26  @mode:--
.id:14  .type:thunderChc       @time:21-30  @mode:--
.id:15  .type:thunderChc       @time:26-30  @mode:--
.id:16  .type:rainChc          @time:17-30  @mode:--
.id:17  .type:rainChc          @time:17-21  @mode:--
.id:18  .type:rainChc          @time:17-26  @mode:--
.id:19  .type:rainChc          @time:21-30  @mode:--
.id:20  .type:rainChc          @time:26-30  @mode:--
.id:21  .type:snowChc          @time:17-30  @mode:--
.id:22  .type:snowChc          @time:17-21  @mode:--
.id:23  .type:snowChc          @time:17-26  @mode:--
.id:24  .type:snowChc          @time:21-30  @mode:--
.id:25  .type:snowChc          @time:26-30  @mode:--
.id:26  .type:freezeRainChc    @time:17-30  @mode:--
.id:27  .type:freezeRainChc    @time:17-21  @mode:--
.id:28  .type:freezeRainChc    @time:17-26  @mode:--
.id:29  .type:freezeRainChc    @time:21-30  @mode:--
.id:30  .type:freezeRainChc    @time:26-30  @mode:--
.id:31  .type:sleetChc         @time:17-30  @mode:--
.id:32  .type:sleetChc         @time:17-21  @mode:--
.id:33  .type:sleetChc         @time:17-26  @mode:--
.id:34  .type:sleetChc         @time:21-30  @mode:--
.id:35  .type:sleetChc         @time:26-30  @mode:--
```

Partly cloudy ,	5
with a low around 56 .	0
South southwest wind around 10 mph .	3 2

[Example 2]

```
.id:0    .type:temperature      @time:6-21    #min:58        #mean:63
                                                             #max:67
.id:1    .type:windChill        @time:6-21    #min:0         #mean:0
                                                             #max:0
.id:2    .type:windSpeed        @time:6-21    #min:10        #mean:19
                                              #max:23        @bucket:10-20
.id:3    .type:windDir          @time:6-21    @mode:S
.id:4    .type:gust             @time:6-21    #min:0         #mean:24
                                                             #max:31
.id:5    .type:skyCover         @time:6-21    @bucket:50-75
.id:6    .type:skyCover         @time:6-9     @bucket:25-50
.id:7    .type:skyCover         @time:6-13    @bucket:25-50
.id:8    .type:skyCover         @time:6-13    @bucket:50-75
.id:9    .type:skyCover         @time:13-21   @bucket:50-75
.id:10   .type:precipPotential  @time:6-21    #min:10        #mean:24
                                                             #max:36
.id:11   .type:thunderChc       @time:6-21    @mode:Chc
.id:12   .type:thunderChc       @time:6-9     @mode:--
.id:13   .type:thunderChc       @time:6-13    @mode:--
.id:14   .type:thunderChc       @time:6-13    @mode:Chc
.id:15   .type:thunderChc       @time:13-21   @mode:Chc
.id:16   .type:rainChc          @time:6-21    @mode:Chc
.id:17   .type:rainChc          @time:6-9     @mode:--
.id:18   .type:rainChc          @time:6-13    @mode:--
.id:19   .type:rainChc          @time:6-13    @mode:Chc
.id:20   .type:rainChc          @time:13-21   @mode:Chc
.id:21   .type:snowChc          @time:6-21    @mode:--
.id:22   .type:snowChc          @time:6-9     @mode:--
.id:23   .type:snowChc          @time:6-13    @mode:--
.id:24   .type:snowChc          @time:6-13    @mode:--
.id:25   .type:snowChc          @time:13-21   @mode:--
.id:26   .type:freezeRainChc    @time:6-21    @mode:--
.id:27   .type:freezeRainChc    @time:6-9     @mode:--
.id:28   .type:freezeRainChc    @time:6-13    @mode:--
.id:29   .type:freezeRainChc    @time:6-13    @mode:--
.id:30   .type:freezeRainChc    @time:13-21   @mode:--
.id:31   .type:sleetChc         @time:6-21    @mode:--
.id:32   .type:sleetChc         @time:6-9     @mode:--
.id:33   .type:sleetChc         @time:6-13    @mode:--
.id:34   .type:sleetChc         @time:6-13    @mode:--
.id:35   .type:sleetChc         @time:13-21   @mode:--
```

A 30 percent chance of showers and thunderstorms after noon .	10 20 15
Mostly cloudy ,	5
with a high near 69 .	0
South wind between 10 and 20 mph ,	3 2
with gusts as high as 30 mph .	4

[Example 3]

```
.id:0   .type:temperature      @time:17-30  #min:53         #mean:61
                                                            #max:73
.id:1   .type:windChill        @time:17-30  #min:0          #mean:0
                                                            #max:0
.id:2   .type:windSpeed        @time:17-30  #min:23         #mean:24
                                            #max:26         @bucket:10-20
.id:3   .type:windDir          @time:17-30  @mode:WSW
.id:4   .type:gust             @time:17-30  #min:31         #mean:33
                                                            #max:36
.id:5   .type:skyCover         @time:17-30  @bucket:75-100
.id:6   .type:skyCover         @time:17-21  @bucket:75-100
.id:7   .type:skyCover         @time:17-26  @bucket:75-100
.id:8   .type:skyCover         @time:21-30  @bucket:25-50
.id:9   .type:skyCover         @time:26-30  @bucket:25-50
.id:10  .type:precipPotential  @time:17-30  #min:1          #mean:42
                                                            #max:70
.id:11  .type:thunderChc       @time:17-30  @mode:SChc
.id:12  .type:thunderChc       @time:17-21  @mode:Lkly
.id:13  .type:thunderChc       @time:17-26  @mode:Lkly
.id:14  .type:thunderChc       @time:21-30  @mode:SChc
.id:15  .type:thunderChc       @time:26-30  @mode:SChc
.id:16  .type:rainChc          @time:17-30  @mode:SChc
.id:17  .type:rainChc          @time:17-21  @mode:Lkly
.id:18  .type:rainChc          @time:17-26  @mode:Lkly
.id:19  .type:rainChc          @time:21-30  @mode:SChc
.id:20  .type:rainChc          @time:26-30  @mode:SChc
.id:21  .type:snowChc          @time:17-30  @mode:--
.id:22  .type:snowChc          @time:17-21  @mode:--
.id:23  .type:snowChc          @time:17-26  @mode:--
.id:24  .type:snowChc          @time:21-30  @mode:--
.id:25  .type:snowChc          @time:26-30  @mode:--
.id:26  .type:freezeRainChc    @time:17-30  @mode:--
.id:27  .type:freezeRainChc    @time:17-21  @mode:--
.id:28  .type:freezeRainChc    @time:17-26  @mode:--
.id:29  .type:freezeRainChc    @time:21-30  @mode:--
.id:30  .type:freezeRainChc    @time:26-30  @mode:--
.id:31  .type:sleetChc         @time:17-30  @mode:--
.id:32  .type:sleetChc         @time:17-21  @mode:--
.id:33  .type:sleetChc         @time:17-26  @mode:--
.id:34  .type:sleetChc         @time:21-30  @mode:--
.id:35  .type:sleetChc         @time:26-30  @mode:--
```

Showers and thunderstorms likely ,	18 13
mainly before midnight .	18 13
Some of the storms could be severe .	–
Cloudy ,	5
then gradually becoming partly cloudy ,	5
with a low around 51 .	0
Windy ,	2
with a southwest wind around 25 mph ,	3 2
with gusts as high as 35 mph .	4
Chance of precipitation is 70 % .	10

[Example 4]

```
.id:0   .type:temperature      @time:6-21   #min:29         #mean:33
                                                            #max:38
.id:1   .type:windChill        @time:6-21   #min:0          #mean:0
                                                            #max:0
.id:2   .type:windSpeed        @time:6-21   #min:7          #mean:10
                                            #max:15         @bucket:10-20
.id:3   .type:windDir          @time:6-21   @mode:S
.id:4   .type:gust             @time:6-21   #min:0          #mean:24
                                                            #max:31
.id:5   .type:skyCover         @time:6-21   @bucket:75-100
.id:6   .type:skyCover         @time:6-9    @bucket:75-100
.id:7   .type:skyCover         @time:6-13   @bucket:75-100
.id:8   .type:skyCover         @time:6-13   @bucket:75-100
.id:9   .type:skyCover         @time:13-21  @bucket:75-100
.id:10  .type:precipPotential  @time:6-21   #min:57         #mean:73
                                                            #max:91
.id:11  .type:thunderChc       @time:6-21   @mode:--
.id:12  .type:thunderChc       @time:6-9    @mode:--
.id:13  .type:thunderChc       @time:6-13   @mode:--
.id:14  .type:thunderChc       @time:6-13   @mode:--
.id:15  .type:thunderChc       @time:13-21  @mode:--
.id:16  .type:rainChc          @time:6-21   @mode:Lkly
.id:17  .type:rainChc          @time:6-9    @mode:Lkly
.id:18  .type:rainChc          @time:6-13   @mode:--
.id:19  .type:rainChc          @time:6-13   @mode:--
.id:20  .type:rainChc          @time:13-21  @mode:--
.id:21  .type:snowChc          @time:6-21   @mode:Lkly
.id:22  .type:snowChc          @time:6-9    @mode:Lkly
.id:23  .type:snowChc          @time:6-13   @mode:Lkly
.id:24  .type:snowChc          @time:6-13   @mode:Lkly
.id:25  .type:snowChc          @time:13-21  @mode:Lkly
.id:26  .type:freezeRainChc    @time:6-21   @mode:--
.id:27  .type:freezeRainChc    @time:6-9    @mode:--
.id:28  .type:freezeRainChc    @time:6-13   @mode:--
.id:29  .type:freezeRainChc    @time:6-13   @mode:--
.id:30  .type:freezeRainChc    @time:13-21  @mode:--
.id:31  .type:sleetChc         @time:6-21   @mode:--
.id:32  .type:sleetChc         @time:6-9    @mode:--
.id:33  .type:sleetChc         @time:6-13   @mode:--
.id:34  .type:sleetChc         @time:6-13   @mode:--
.id:35  .type:sleetChc         @time:13-21  @mode:--
```

Rain and snow likely before 11am ,	17 22
then snow .	21
High near 38 .	0
South wind between 8 and 15 mph .	3 2
Chance of precipitation is 90 % .	10
New snow accumulation of 1 to 3 inches possible .	21

A.3 ATIS Examples

[Example 1]

```
.id:0  .type:city     @from:any       @location:--  @services:--
                                      @stop:--      @to:--
.id:1  .type:flight   @air_code:--    @airline:--   @class_type:--
                      @direction:--   @engine:--    @fare:--
                      @fl_number:--   @from:any     @manufacturer:--
                      @price:--       @stop:--      @to:mke
                                                    @year:--
.id:2  .type:search   @of:--          @typed:lambda @what:flight
```

list all the flights that arrive at general mitchell international from various cities	0 1 2

[Example 2]

```
.id:0  .type:flight   @air_code:--    @airline:--      @class_type:--
                      @direction:--   @engine:--       @fare:--
                      @fl_number:--   @from:pittsburgh @manufacturer:--
                      @price:--       @stop:--         @to:boston
                                                       @year:--
.id:1  .type:search   @of:--          @typed:lambda    @what:flight
.id:2  .type:day      @day:saturday   @dep_ar_ret:dep
```

pittsburgh to boston saturday	0 1 2

[Example 3]

```
.id:0  .type:flight       @air_code:--      @airline:ua        @class_type:--
                          @direction:--     @engine:--         @fare:--
                          @fl_number:--     @from:pittsburgh   @manufacturer:--
                          @price:--         @stop:denver       @to:francisco
                                                               @year:--
.id:1  .type:search       @of:--            @typed:lambda      @what:flight
.id:2  .type:when         @dep-ar:dep       @when:morning
.id:3  .type:day_number   @day_number:20    @dep_ar_ret:dep
.id:4  .type:month        @dep_ar_ret:dep   @month:september
```

what flights do you have in the morning of september twentieth on united airlines from pittsburgh to san francisco and a stopover in denver	0 1 2 3 4

[Example 4]

```
.id:0  .type:flight   @air_code:--           @airline:--         @class_type:--
                      @direction:--          @engine:--          @fare:--
                      @fl_number:--          @from:washington    @manufacturer:--
                      @price:--              @stop:--            @to:boston
.id:1  .type:search   @of:departure_time     @typed:argmax       @what:flight
```

what is the last flight from washington to boston	0 1

A.4 WINHELP Examples

[Example 1]

```
.id:0  .type:nav-desktop       @envCmd:left click $objName:start
                                                  @objType:Button
.id:1  .type:nav-start         @envCmd:left click $objName:settings
                                                  @objType:Button
.id:2  .type:nav-start-target  @envCmd:left click $objName:control panel
                                                  @objType:Button
.id:3  .type:nav-contMenu      @envCmd:left click $objName:view
                                                  @objType:Button
.id:4  .type:action-contMenu   @envCmd:left click $objName:large icons
                               $typeInto:--       @objType:Menu
.id:5  .type:nav-contMenu      @envCmd:left click $objName:view
                                                  @objType:Button
.id:6  .type:action-contMenu   @envCmd:left click $objName:small icons
                               $typeInto:--       @objType:Menu
```

click start , point to settings , and then click control panel .	0 1 2
on the view menu , click large icons .	3 4
on the view menu , click small icons .	5 6

[Example 2]

```
.id:0  .type:nav-desk-target  @envCmd:double click $objName:my computer
                                                   @objType:Item
.id:1  .type:nav-win          @envCmd:left click   $objName:tools
                                                   @objType:Button
.id:2  .type:nav-win-target   @envCmd:left click   $objName:folder
                                                   options
                                                   @objType:Menu
.id:3  .type:nav-contMenu     @envCmd:left click   $objName:file types
                                                   @objType:Tab
.id:4  .type:nav-location     $name:registered     @objType:Window
                              file types
.id:5  .type:nav-contMenu     @envCmd:left click   @objType:Item
       $objName:xls microsoft excel worksheet
.id:6  .type:nav-contMenu-    @envCmd:left click   $objName:advanced
       -target                                     @objType:Button
.id:7  .type:action-contMenu  @envCmd:left click   $typeInto:--
       $objName:confirm open after download        @objType:checkbox
.id:8  .type:exit-contMenu    @envCmd:left click   $objName:ok
                                                   @objType:Button
```

double-click my computer .	0
on the tools menu , click folder options .	1 2
click the file types tab .	3
under registered file types , click to select xls microsoft excel worksheet , and then click advanced .	4 5 6
click to clear the confirm open after download check box , and then click ok .	7 8

[Example 3]

```
.id:0  .type:nav-desktop      @envCmd:left click  $objName:start
                                                  @objType:Button
.id:1  .type:nav-start        @envCmd:left click  $objName:programs
                                                  @objType:Button
.id:2  .type:nav-start        @envCmd:left click  $objName:accessories
                                                  @objType:Button
.id:3  .type:nav-start        @envCmd:left click  $objName:accessibility
                                                  @objType:Button
.id:4  .type:nav-start-target @envCmd:left click  $objName:magnifier
                                                  @objType:Button
.id:5  .type:action-contMenu  @envCmd:left click  $typeInto:--
       $objName:follow mouse cursor               @objType:checkbox
.id:6  .type:exit-contMenu    @envCmd:left click  $objName:exit
                                                  @objType:Button
```

click start , point to programs , point to accessories , point to accessibility , and then click magnifier .	0 1 2 3 4
click the follow mouse cursor check box to select it , and then click exit .	5 6

[Example 4]

```
.id:0.type:nav-desk          @envCmd:left click   $objName:start
                                                  @objType:Button
.id:1.type:nav-start-target  @envCmd:left click   $objName:help
                                                  @objType:Button
.id:2.type:nav-contMenu      @envCmd:left click   $objName:search
                                                  @objType:Tab
.id:3.type:action-contMenu   @envCmd:type_into    @objType:Edit
      $objName:type in the keyword to find
      $typeInto:internet connection sharing
.id:4.type:nav-contMenu      @envCmd:left click   $objName:list topics
                                                  @objType:Button
.id:5.type:nav-contMenu-tar  @envCmd:double click @objType:Item
      $objName:internet connection sharing
```

click start , and then click help .	0 1
on the search tab , type internet connection sharing in the type in the keyword to find box , and then click list topics .	2 3 4
double-click internet connection sharing in the select topic box .	5 6

Appendix B

Document Planning CFG Rules

In the following sections we give a list of document planning rules based on rule (1) of grammar G_{GEN}++ (Table 5.1) for WEATHERGOV and WINHELP. The rules were extracted using both unsupervised (Section 5.4.1) and domain-based automatic alignments (Section 5.4.3) and were then binarized; in the case of WINHELP, the rules were also horizontally markovised. We refer to each set of rules using the names of the corresponding systems they were used on, namely DP-UNSUP and DP-AUTO. We present the top-20 scoring rules for each domain rooted on the non-terminal D.

B.1 WeatherGov Document Plan Rules

B.1.1 DP-Unsup

```
D → SENT(skyCover, temperature) SENT(windDir, windSpeed)
D → SENT(rainChance) [SENT(skyCover, temperature)
SENT(windDir, windSpeed) SENT(precipPotential)]
D → SENT(skyCover, temperature) SENT(windDir, windSpeed, gust)
D → SENT(precipPotential, rainChance) [SENT(skyCover, temperature)
SENT(windDir, windSpeed)]
D → SENT(rainChance) [SENT(skyCover, temperature)
SENT(windDir, windSpeed) SENT(precipPotential, rainChance)]
D → SENT(skyCover, temperature) [SENT(windSpeed, windDir, gust)]
D → SENT(skyCover, temperature) SENT(windSpeed)]
D → SENT(precipPotential, rainChance) [SENT(skyCover, temperature)
SENT(windDir, windSpeed, gust)]
D → SENT(rainChance) [SENT(skyCover, temperature)
SENT(windSpeed, windDir, gust) SENT(precipPotential)]
D → SENT(rainChance temperature, windDir, windSpeed)
SENT(precipPotential)
D → SENT(skyCover, temperature) SENT(windSpeed, windDir)
D → SENT(precipPotential, rainChance) [SENT(skyCover, temperature)
SENT(windSpeed, windDir, gust)]
D → SENT(rainChance) [SENT(skyCover, temperature)
SENT(windDir, windSpeed) SENT(gust) SENT(precipPotential)]
D → SENT(rainChance) [SENT(skyCover, temperature) SENT(windSpeed)
SENT(precipPotential)]
D → SENT(rainChance) [SENT(temperature, windDir, windSpeed) SENT(gust)
SENT(precipPotential)]
D → SENT(rainChance, snowChance) [SENT(skyCover, temperature)
SENT(windDir, windSpeed) SENT(precipPotential, snowChance)]
D → SENT(rainChance) [SENT(skyCover, temperature)
SENT(windDir, windSpeed)]
D → SENT(rainChance, thunderChance) [SENT(temperature, windDir)
SENT(windSpeed gust) SENT(precipPotential)]
D → SENT(rainChance, thunderChance) [SENT(temperature)
SENT(windSpeed, windDir) SENT(gust) SENT(precipPotential)
SENT(rainChance)]
D → SENT(rainChance, thunderChance) [SENT(skyCover, temperature)
SENT(windDir, windSpeed) SENT(precipPotential) SENT(rainChance,
thunderChance)]
```

B.1.2 DP-Auto

```
D → SENT(skyCover, temperature) SENT(windDir, windSpeed)
D → SENT(rainChance) SENT(skyCover, temperature)
SENT(windDir, windSpeed) SENT(precipPotential)]
D → SENT(skyCover, temperature) SENT(windDir, windSpeed, gust)
D → SENT(precipPotential, rainChance) [SENT(skyCover, temperature)
SENT(windDir, windSpeed)]
D → SENT(rainChance) [SENT(skyCover, temperature)
SENT(windDir, windSpeed) SENT(precipPotential, rainChance)]
D → SENT(skyCover, temperature) [SENT(windSpeed, windDir) SENT(gust)]
D → SENT(skyCover, temperature) SENT(windSpeed)
D → SENT(rainChance) [SENT(skyCover, temperature)
SENT(windDir, windSpeed) SENT(precipPotential)]
D → SENT(precipPotential, rainChance) [SENT(skyCover, temperature)
SENT(windDir, windSpeed, gust)]
D → SENT(rainChance) [SENT(skyCover, temperature)
SENT(windSpeed, windDir) SENT(gust) SENT(precipPotential)]
D → SENT(rainChance)[ SENT(temperature) SENT(windDir, windSpeed)
SENT(precipPotential)]
D → SENT(skyCover, temperature) SENT(windSpeed, windDir)
D → SENT(rainChance) [SENT(skyCover, temperature)
SENT(windDir, windSpeed, gust) SENT(precipPotential)]
D → SENT(precipPotential, rainChance) [SENT(skyCover, temperature)
SENT(windSpeed, windDir) SENT(gust)]
D → SENT(rainChance) [SENT(temperature) SENT(windDir, windSpeed, gust)
SENT(precipPotential)]
D → SENT(rainChance, snowChance) [SENT(skyCover, temperature)
SENT(windDir, windSpeed) SENT(precipPotential, snowChance)]
D → SENT(rainChance, thunderChance) [SENT(temperature)
SENT(windDir, windSpeed) SENT(gust, precipPotential)]
D → SENT(rainChance, thunderChance) [SENT(skyCover, temperature)
SENT(windDir, windSpeed) SENT(precipPotential, rainChance, thunderChance)]
D → SENT(rainChance) [SENT(skyCover, temperature)
SENT(windDir, windSpeed)]
D → SENT(rainChance, thunderChance) [SENT(temperature)
SENT(windSpeed, windDir, gust) SENT(precipPotential, rainChance)]
```

B.2 WinHelp Document Plan Rules

B.2.1 DP-Unsup

```
D → SENT(nav-desk, nav-start, nav-start-target)
⟨D...SENT(nav-desk, nav-start, nav-start-target)⟩
D → SENT(nav-desk, nav-start, nav-start-target, nav-win-target)
⟨D...SENT(nav-desk, nav-start, nav-start-target, nav-win-target)⟩
D → SENT(nav-desk-target) ⟨D...SENT(nav-desk-target)⟩
D → SENT(nav-desk, nav-win-target) ⟨D...SENT(nav-desk, nav-win-target)⟩
D → SENT(nav-desk, nav-desk-target) ⟨D...SENT(nav-desk, nav-desk-target)⟩
D → SENT(nav-win-target) ⟨D...SENT(nav-win-target)⟩
D → SENT(nav-desk, exit-contMenu) ⟨D...SENT(nav-desk, exit-contMenu)⟩
D → SENT(nav-win, nav-win-target) ⟨D...SENT(nav-win, nav-win-target)⟩
D → SENT(nav-start, nav-start-target) ⟨D...SENT(nav-start,
nav-start-target)⟩
D → SENT(nav-desk, nav-win-target, nav-desk-target)
⟨D...SENT(nav-desk, nav-win-target, nav-desk-target)⟩
D → SENT(nav-desk, nav-start, nav-win-target) ⟨D...SENT(nav-desk,
nav-start, nav-win-target)⟩
D → SENT(nav-desk, nav-start, nav-start-target) SENT(nav-win,
nav-win-target)
D → SENT(nav-desk, nav-start, nav-start-target) SENT(action-contMenu,
exit-contMenu)
D → SENT(nav-desk, nav-start, action-contMenu, nav-win-target)
⟨D...SENT(nav-desk, nav-start, action-contMenu, nav-win-target)⟩
D → SENT(nav-desk, nav-start-target) ⟨D...SENT(nav-desk,
nav-start-target)⟩
D → SENT(nav-desk, nav-start) ⟨D...SENT(nav-desk, nav-start)⟩
D → SENT(nav-desk-target, nav-win-target) ⟨D...SENT(nav-desk-target,
nav-win-target)⟩
D → SENT(action-contMenu) ⟨D...SENT(action-contMenu)⟩
D → SENT(nav-win, nav-win-target, nav-desk-target)
⟨D...SENT(nav-win, nav-win-target, nav-desk-target)⟩
D → SENT(nav-win, nav-win-target) SENT(nav-contMenu, action-contMenu,
exit-contMenu)
```

B.2.2 DP-AUTO

```
D → SENT(nav-desk, nav-start, nav-start-target)
⟨D...SENT(nav-desk, nav-start, nav-start-target)⟩
D → SENT(nav-desk, nav-desk-target) ⟨D...SENT(nav-desk, nav-desk-target)⟩
D → SENT(nav-desk, nav-start, nav-start-target, nav-win-target)
⟨D...SENT(nav-desk, nav-start, nav-start-target, nav-win-target)⟩
D → SENT(nav-desk-targe)t ⟨D...SENT(SENT-nav-desk-target)⟩
D → SENT(nav-win, nav-win-target) ⟨D...SENT(nav-win, nav-win-target)⟩
D → SENT(nav-location, nav-program, nav-program-target)
⟨D...SENT(nav-location, nav-program, nav-program-target)⟩
D → SENT(SENT-nav-win-target) ⟨D...SENT(SENT-nav-win-target)⟩
D → SENT(nav-program, nav-program-target, nav-contMenu)
⟨D...SENT(nav-program, nav-program-target, nav-contMenu)⟩
D → SENT(nav-location, nav-desk, nav-desk-target)
⟨D...SENT(nav-location, nav-desk, nav-desk-target)⟩
D → SENT(nav-desk, nav-start, nav-start-target, nav-win-target)
SENT(nav-contMenu, action-contMenu)
D → SENT(nav-desk, nav-start, nav-start-target) SENT(nav-win,
nav-win-target)
D → SENT(nav-desk, nav-start, nav-start-target) SENT(action-contMenu,
exit-contMenu)
D → SENT(nav-desk, nav-start) ⟨D...SENT(nav-desk, nav-start)⟩
D → SENT(nav-desk-target, nav-win-target) ⟨D...SENT(nav-desk-target,
nav-win-target)⟩
D → SENT(nav-desk) ⟨D...SENT(SENT-nav-desk)⟩
D → SENT(nav-win, nav-win-target) SENT(nav-contMenu, action-contMenu,
exit-contMenu)
D → SENT(nav-win, nav-win-target) SENT(nav-contMenu, action-contMenu)
D → SENT(nav-win, nav-win-target) SENT(action-contMenu, exit-contMenu)
D → SENT(nav-win, nav-win-target) SENT(action-contMenu)
D → SENT(nav-start, nav-start-target) ⟨D...SENT(nav-start,
nav-start-target)⟩
```

Appendix C

Experimental Instructions

In the following sections we give the experimental instructions we supplied the human judges with, prior to conducting the human evaluation experiments presented in Sections 4.5 and 5.5. The experiments were performed over the Internet using Amazon Mechanical Turk (`www.mturk.com`). The experiment instructions includ an introduction to the task with some background information for the domain, followed by rating examples and guidelines. In the end we also included some procedural information and further guidance on acquiring personal details. Sections C.1-C.4 correspond to the experiments evaluating the models presented in Chapter 4 for ROBOCUP, WEATHERGOV, ATIS, and WINHELP, across two dimensions, namely fluency and semantic correctness. Section C.5 corresponds to the experiments evaluating the models presented in Chapter 5 for WINHELP only, across three dimensions, namely, fluency, semantic correctness, and coherence. Finally, we give the procedural and personal details instructions in Section C.1 only, for the sake of brevity.

C.1 ROBOCUP instructions

Instructions

In this experiment you will be given tables that contain some facts about a soccer game and their translation in natural language. For instance, Example 1 below describes an action that passes the ball (see the column labelled as Category in the table) who is performing it (i.e., `pink3`; see Field *Actor*) and who is the Recipient (i.e., `pink7`). Here the table is translated as: *Pink3 kicks out to Pink7.*

Example 1

Category	Fields	
Pass	Actor: `pink3`	Recipient: `pink7`
Pink3 kicks out to Pink7.		

Typically, tables will describe a single action (e.g., **Pass**, **Kick**) accompanied with a few fields (e.g., *Actor*, *Recipient* in Example 1) and their values (e.g., `pink3`, `pink7`). Some events may not have fields.

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 actions, 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 the translation might paraphrase what is mentioned in the table. Example `pink7` in the table might be translated as *pinkie7*. Such slight divergences are normal and should not be penalized.

Rating Examples

n Example 1 you would probably give the translation a high score for Fluency, 4-5, since it is coherent and does not contain any grammatical errors. However, you would probably give it a mid-range score for Semantic Correctness (e.g., 3 or 4) because the table describes a passing event whereas the translation mentions a kicking event (i.e., the two events are not the same).

Example 2

Category	Fields	
Kick	Actor: `purple7`	Recipient: `purple5`
Purple7 kicks the ball out to Purple5.		

In example 2 you should give the translation high scores for both dimensions (e.g., 4 or 5), Fluency and Semantic Correctness. The text is grammatical and describes the content of the table accurately. 4-5 would be good scores.

Example 3

Category	Fields	
Pass	Actor: `purple3`	Recipient: `purple5`
To Purple5. To Purple5.		

In example 3 the translation is neither fluent nor semantically correct. So you would probably give it a low score on both dimensions (e.g., 1 or 2). The text is repetitive and not very descriptive of the content of the table. The phrase *to Purple5* probably corresponds to the Recipient field with the value `purple5` but it is not clear from the translation who is doing what to `purple5`. 1-2 are the appropriate scores for both dimensions.

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 analysing 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 judgement when rating each translation.

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.

C.2 WeatherGov Instructions

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: `17-06(+1 day)`	min: `30`	mean: `40` max: `47`
Wind Direction	time: `17-06(+1 day)`	mode: `SE`	
Cloud Sky Cover	time: `17-06(+1 day)`	percent: `25--50`	
Chance of Rain	time: `17-21`	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 `17-06(+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').

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: 17-06(+1 day)	min: 40	mean: 45	max: 50
Wind Direction	time: 17-06(+1 day)	mode: S		
Wind Speed	time: 17-06(+1 day)	min: 5	mean: 7	max: 15
Cloud Sky Cover	time: 17-06(+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: `17-06(+1 day)` min: `30` mean: `40` max:`47`
Wind Direction	time: `17-06(+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 penalise 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.

C.3 ATIS Instructions

Instructions

In this experiment you will be given tables that contain some facts about booking flights, or other related information (e.g., airline codes, booking code fares and so on) and their translation in natural language. All translations are hypothetical responses to a telephone air-travel booking system. Example 1 below describes a query to book a flight (see the second line that has the category **Query**) from New York to Seattle (see the first line with category **Flight Info**). Here the table is translated as: *Show me the flights from New York going to Seattle.*

Typically, tables will describe partial facts that fall under different categories and constitute together a booking or other flight-related scenario. Each fact belongs to a category (e.g., **Flight Info**, **Query**), accompanied with a few fields and their values.

The values can contain either words (e.g., `show` for the **Query** category), or codes related to cities or airports (e.g., `new_york` or `seattle` for the `Flight Info` category).

More precisely, the field type *show* in the **Query** category, corresponds to asking for specific information. The information that we are looking for is related to a flight or in general flights (field *what*).

Example 1

Category	Fields	
✈ **Flight Info**	from: `new_york`	to: `seattle`
🔍 **Query**	type: `show`	what: `flight`
Show me the flights from New York going to Seattle		

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 the translation might expand and paraphrase what is mentioned in the table. For example the field-value pair type: show in the table might be translated as '*show me*' or '*please give me*' or just '*give*'. Such slight divergences are normal and should not be penalized.

Rating Examples

In Example 1 you should give the translation high scores on both dimensions, namely Fluency and Semantic Correctness. The text is grammatical and it expresses the same meaning (i.e., content) as the corresponding table. Here 4-5 would be appropriate numbers.

Example 2

Category	Fields	
✈ **Flight Info**	from: `berlin`	to: `edinburgh`
🔍 **Query**	type: `show`	what: `flight`
📅 **Day**	day: `monday`	dep/ar/ret: `arrival`
🕐 **When**	dep/ar: `arrival`	when: `afternoon`
Show me the flights from Berling going to Edinburgh on Monday		

In Example 2 you would probably give the translation a high score for Fluency (e.g. 4 or 5), since it does not contain any grammatical errors. However, you should give it a low score for Semantic Correctness, (e.g. 1 or 2), because it is missing information, namely that the arrival at Edinburgh is in the afternoon (**When** category).

Example 3

Category	Fields	
✈ **Flight Info**	from: `athens`	to: `london`
🔍 **Query**	type: `show`	what: `flight`
What what what flights Athens London		

In example 3 the translation scores poorly on Fluency and Semantic Correctness. The text has many repetitions, and has grammatical errors. In addition, there is no clear correspondence between the translation and the table. `Athens` probably refers to the departing airport, and `London` to the destination but it is not made explicit in the translation text. Here 1 or 2 would be appropriate scores for both dimensions.

Finally, while judging the translation pay particular attention to the **Query** category. You should penalise translations that appear grammatical and semantically sound but do not verbalize the **Query** category if the latter is present in the table.

C.4 WINHELP Instructions

Instructions

In this experiment you will be given tables that contain instructions on how to complete a task on a Windows 2000 desktop environment (e.g., navigate the start menu, double-click on a button, etc) and their translation in natural language. Typical examples include trouble shooting, such as opening the device manager and enabling a specific parameter, or navigating the internet options menu of the internet explorer. Example 1 below gives step-by-step instructions on how to open the device manager window from the start menu (**Navigate in Desktop**, **Navigate in Start Menu**, **Select in Start Menu**, **Select in Window**, **Navigate in Context Menu**, **Final Goal**). Here the table is translated as: *Click start, point to settings, click control panel. Double-click system, on the hardware tab, click device manager.*

Example 1

Category	Fields		
Navigate in Desktop	How: `left-click`	Target: `start`	Type: `Button`
Navigate in Start Menu	How: `left-click`	Target: `settings`	Type: `Button`
Select in Start Menu	How: `left-click`	Target: `ctrl panel`	Type: `Button`
Select in Start Menu	How: `left-click`	Target: `system`	Type: `Item`
Nav. in Context Menu	How: `left-click`	Target: `hardware`	Type: `Tab`
Final Goal	How: `left-click`	Target: `device manager`	Type: `Button`
Click start, point to settings, click control panel. Double-click system, in the Internet Explorer Menu, on the hardware tab, click device manager.			

Each row in the table contains a different instruction. The first column describes in words the different types of instructions (e.g., **Select in Start Menu**, **Final Goal**). The following columns contain fields and their values. These give the exact details on how to realize the instruction. For example the instruction **Navigate on Desktop** has three fields, *How*, *Target* and *Type*. The values for these fields are `left-click`, `start`, and `Button`. This can be simply interpreted as left-clicking on the start button, which is found on the desktop. The instruction **Select in Window** has the values `left-click`, `system`, and `item`. It can be interpreted as left-clicking on an item named ‘system’, found in the window, most probably opened in the previous step. Field values can be either words (e.g., `Item` for the field *Type*), or strings (e.g., `device manager` or `system` for the field *Target*). For example the second row of the table with category **Navigate in Start Menu** can be described as follows: on the menu under the start button opened from the previous step, we go to `settings` (field *Target*) which is a click-able button (field *Type*) in the menu and `left-click` on it (field *How*). The next row with the instruction **Select in Start Menu**, continues from the previous step as follows: we `left-click` (field *How*) on the `control panel` (field *Target*) `button` (field *Type*).

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 instructions (i.e., 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.

Rating Examples

In Example 1 you would probably give the translation a high score for Fluency (e.g., 4-5), since it is in well-formed English 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, '*Internet Explorer*' and '*options menu*' possibly under the internet explorer, are never mentioned in the table of instructions. **Bare in mind** that troubleshooting guides should be fairly specific. A good step-by-step guide should not leave space for ambiguity, mixed interpretations or omit any instruction steps mentioned in the table.

Example 2

<table>
<tr><th>Category</th><th colspan="3">Fields</th></tr>
<tr><td><b>Navigate in Window</b></td><td>How: <code>dbl-click</code></td><td>Target: <code>user configuration</code></td><td>Type: <code>Tree Item</code></td></tr>
<tr><td><b>Navigate in Window</b></td><td>How: <code>dbl-click</code></td><td>Target: <code>administrative templates</code></td><td>Type: <code>Item</code></td></tr>
<tr><td><b>Nav. in Context Menu</b></td><td>How: <code>dbl-click</code></td><td>Target: <code>system</code></td><td>Type: <code>Item</code></td></tr>
<tr><td><b>Slct in Context Menu</b></td><td>How: <code>dbl-click</code></td><td>Target: <code>group policy</code></td><td>Type: <code>Item</code></td></tr>
<tr><td><b>Nav. in Context Menu</b></td><td>How: <code>dbl-click</code></td><td>Target: <code>automatic updates to .adm files</code></td><td>Type: <code>Tab</code></td></tr>
<tr><td><b>Final Goal</b></td><td>How: <code>left-click</code></td><td>Target: <code>enabled</code></td><td>Type: <code>Radio</code></td></tr>
<tr><td colspan="4">Double-click user configuration, and then double-click administrative templates.
Double-click system, and then double-click group policy.
Double-click automatic update to .adm files, and click enabled.</td></tr>
</table>

In example 2 you should give the translation high scores on both dimensions, namely Fluency and Semantic Correctness. The text is grammatical and it captures the content of the table without any omissions In other words, the text is readily comprehensible and the information described is well organized and flows from one step to the next in a logical fashion.

Example 3

<table>
<tr><th>Category</th><th colspan="3">Fields</th></tr>
<tr><td><b>Navigate in Desktop</b></td><td>How: <code>left-click</code></td><td>Target: <code>start</code></td><td>Type: <code>Button</code></td></tr>
<tr><td><b>Navigate in Start Menu</b></td><td>How: <code>left-click</code></td><td>Target: <code>settings</code></td><td>Type: <code>Button</code></td></tr>
<tr><td><b>Select in Start Menu</b></td><td>How: <code>left-click</code></td><td>Target: <code>ctrl panel</code></td><td>Type: <code>Button</code></td></tr>
<tr><td><b>Nav. in Context Menu</b></td><td>How: <code>left-click</code></td><td>Target: <code>view</code></td><td>Type: <code>Button</code></td></tr>
<tr><td><b>Final Goal</b></td><td>How: <code>left-click</code></td><td>Target: <code>large icons</code></td><td>Type: <code>Menu</code></td></tr>
<tr><td colspan="4">Click start, click start, and then control panel.
On the view menu, click large icons.
On the view menu, click large icons.</td></tr>
</table>

In example 3 the translation scores poorly across both dimensions, i.e., Fluency and Semantic Correctness. The reader cannot work out which parts of the table are expressed in the text which has many repetitions and is badly organized as a whole. The phrase '*and then control panel*' is missing a verb and causes ambiguity for the user: should they click on the item 'control panel' or hover over it? So, 1-2 would be appropriate scores for both dimensions.

C.5 WINHELP Instructions (with Coherence)

Instructions

In this experiment you will be given tables that contain instructions on how to complete a task on a Windows 2000 desktop environment (e.g., navigate the start menu, double-click on a button, etc) and their translation in natural language. Typical examples include trouble shooting, such as opening the device manager and enabling a specific parameter, or navigating the internet options menu of the internet explorer. Example 1 below gives step-by-step instructions on how to open the device manager window from the start menu (**Navigate in Desktop**, **Navigate in Start Menu**, **Select in Start Menu**, **Select in Window**, **Navigate in Context Menu**, **Final Goal**). Here the table is translated as: *Click start, point to settings, click control panel. Double-click system, on the hardware tab, click device manager.*

Example 1

Category	Fields		
Navigate in Desktop	How: `left-click`	Target: `start`	Type: `Button`
Navigate in Start Menu	How: `left-click`	Target: `settings`	Type: `Button`
Select in Start Menu	How: `left-click`	Target: `ctrl panel`	Type: `Button`
Select in Start Menu	How: `left-click`	Target: `system`	Type: `Item`
Nav. in Context Menu	How: `left-click`	Target: `hardware`	Type: `Tab`
Final Goal	How: `left-click`	Target: `device manager`	Type: `Button`
Click start, point to settings, click control panel. Double-click system, in the Internet Explorer Menu, on the hardware tab, click device manager.			

Each row in the table contains a different instruction. The first column describes in words the different types of instructions (e.g., **Select in Start Menu**, **Final Goal**). The following columns contain fields and their values. These give the exact details on how to realize the instruction. For example the instruction **Navigate on Desktop** has three fields, *How*, *Target* and *Type*. The values for these fields are `left-click`, `start`, and `Button`. This can be simply interpreted as left-clicking on the start button, which is found on the desktop. The instruction **Select in Window** has the values `left-click`, `system`, and `item`. It can be interpreted as left-clicking on an item named 'system', found in the window, most probably opened in the previous step. Field values can be either words (e.g., `Item` for the field *Type*), or strings (e.g., `device manager` or `system` for the field *Target*). For example the second row of the table with category **Navigate in Start Menu** can be described as follows: on the menu under the start button opened from the previous step, we go to `settings` (field *Target*) which is a click-able button (field *Type*) in the menu and `left-click` on it (field *How*). The next row with the instruction **Select in Start Menu**, continues from the previous step as follows: we `left-click` (field *How*) on the `control panel` (field *Target*) `button` (field *Type*).

All natural language translations have been generated by a computer program. Your task is to rate the translations on three dimensions, namely Fluency, Semantic Correctness and Coherence 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 instructions (i.e., 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.

Coherence refers to how comprehensible the translation is. If the text is almost impossible to understand, then you should probably give it a low number. If the text is readily comprehensible and does not require any effort on the reader's part, then you should give it a high number.

Rating Examples

In Example 1 you would probably give the translation a high score for Fluency (e.g., 4-5), since it is in well-formed English 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, '*Internet Explorer*' and '*options menu*' possibly under the internet explorer, are never mentioned in the table of instructions. As far as Coherence is concerned you should give the translation a low-moderate score, 1-3. Even though the text is fluent, it is rather hard to follow, especially when reading the second sentence which contains far too many actions, making it hard to follow. In addition, the transition from the phrase '*double-click system*' to the phrase '*Internet Explorer options menu*' is ambiguous: is internet explorer a click-able item in the system menu or an icon in a window that has popped up as a consequence of double-clicking on system? **Bare in mind** that troubleshooting guides should be fairly specific. A good step-by-step guide should not leave space for ambiguity, mixed interpretations or omit any instruction steps mentioned in the table.

Example 2

Category	Fields		
Navigate in Window	How: `dbl-click`	Target: `user configuration`	Type: `Tree Item`
Navigate in Window	How: `dbl-click`	Target: `administrative templates`	Type: `Item`
Nav. in Context Menu	How: `dbl-click`	Target: `system`	Type: `Item`
Slct in Context Menu	How: `dbl-click`	Target: `group policy`	Type: `Item`
Nav. in Context Menu	How: `dbl-click`	Target: `automatic updates to .adm files`	Type: `Tab`
Final Goal	How: `left-click`	Target: `enabled`	Type: `Radio`
Double-click user configuration, and then double-click administrative templates. Double-click system, and then double-click group policy. Double-click automatic update to .adm files, and click enabled.			

In example 2 you should give the translation high scores on both dimensions, namely Fluency, Semantic Correctness, and Coherence. The text is grammatical, it captures the content of the table without any omissions, and is coherent. In other words, the text is readily comprehensible and the information described is well organized and flows from one step to the next in a logical fashion.

Example 3

Category	Fields		
Navigate in Desktop	How: `left-click`	Target: `start`	Type: `Button`
Navigate in Start Menu	How: `left-click`	Target: `settings`	Type: `Button`
Select in Start Menu	How: `left-click`	Target: `ctrl panel`	Type: `Button`
Nav. in Context Menu	How: `left-click`	Target: `view`	Type: `Button`
Final Goal	How: `left-click`	Target: `large icons`	Type: `Menu`
Click start, click start, and then control panel. On the view menu, click large icons. On the view menu, click large icons.			

In example 3 the translation scores poorly across both dimensions, i.e., Fluency, Semantic Correctness and Coherence. The reader cannot work out which parts of the table are expressed in the text which has many repetitions and is badly organized as a whole. The phrase '*and then control panel*' is missing a verb and causes ambiguity for the user: should they click on the item 'control panel' or hover over it? So, 1-2 would be appropriate scores for all dimensions.

Bibliography

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

Appelt, D. E. (1985). Planning English referring expressions. *Artificial Intelligence*, 26(1):1 – 33.

Banerjee, S. and Lavie, A. (2005). METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 65–72, Ann Arbor, Michigan.

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

Barzilay, R. and Lapata, M. (2005a). Collective content selection for concept-to-text generation. In *Proceedings of the HLT/EMNLP*, pages 331–338, Vancouver.

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

Barzilay, R. and Lapata, M. (2006). Aggregation via set partitioning for natural language generation. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 359–366, New York City, USA.

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

Belz, A. and Reiter, E. (2006). Comparing automatic and human evaluation of NLG systems. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics*, pages 313–320, Trento, Italy.

Berg-Kirkpatrick, T., Bouchard-Côté, A., DeNero, J., and Klein, D. (2010). Painless unsupervised learning with features. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pages 582–590, Los Angeles, California.

Branavan, S., Chen, H., Zettlemoyer, L., and Barzilay, R. (2009). Reinforcement learning for mapping instructions to actions. In *Proceedings 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*, pages 82–90, Suntec, Singapore.

Carroll, J., Copestake, A., Flickinger, D., and Poznanski, V. (1999). An efficient chart generator for (semi-)lexicalist grammars. In *Proceedings of the 7th European Workshop on Natural Language Generation (EWNLG'99)*, pages 86–95, Toulouse, France.

Charniak, E. and 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*, pages 173–180, Ann Arbor, Michigan.

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

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

Cohn, T., Blunsom, P., and Goldwater, S. (2010). Inducing tree-substitution grammars. *Journal of Machine Learning Research*, 11(November):3053–3096.

Collins, M. (1999). *Head-Driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania.

Collins, M. (2000). Discriminative reranking for natural language parsing. In *Proceedings of the 17th International Conference on Machine Learning*, pages 175–182, Stanford, California.

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*, pages 1–8, Philadelphia, Pennsylvania.

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

Dale, R. (1988). *Generating referring expressions in a domain of objects and processes*. PhD thesis, University of Edinburgh.

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

Danlos, L. (1984). Conceptual and linguistic decisions in generation. In *Proceedings of the 10th International Conference on Computational Linguistics and 22nd annual* *meeting on Association for Computational Linguistics*, ACL ’84, pages 501–504, Stanford, California.

Davidson, D. (1980). *Essays on Actions and Events*. Clarendon Press, Oxford, second edition.

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

Denkowski, M. and Lavie, A. (2011). Meteor 1.3: Automatic Metric for Reliable Optimization and Evaluation of Machine Translation Systems. In *Proceedings of the EMNLP 2011 Workshop on Statistical Machine Translation*.

Duboue, P. A. and McKeown, K. R. (2001). Empirically estimating order constraints for content planning in generation. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, pages 172–179.

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

Elhadad, M. and Robin, J. (1998). Surge: a comprehensive plug-in syntactic realization component for text generation. Technical report.

Ellis, A. and Young, A. (1996). *Human Cognitive Neuropsychology: A Textbook with Readings*. Psychology Press.

Galanis, D. and Androutsopoulos, I. (2007). Generating multilingual descriptions from linguistically annotated owl ontologies: the naturalowl system. In *Proceedings of the Eleventh European Workshop on Natural Language Generation*, ENLG ’07, pages 143–146, Wadern, Germany.

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

Ge, R. and Mooney, R. J. (2006). Discriminative reranking for semantic parsing. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 263–270, Sydney, Australia.

Goldberg, E., Driedger, N., and 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.

Gouaillier, D., Hugel, V., Blazevic, P., Kilner, C., Monceaux, J., 0002, P. L., Marnier, B., Serre, J., and Maisonnier, B. (2008). The nao humanoid: a combination of performance and affordability. *CoRR*, abs/0807.3223.

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

Grimes, J. (1975). *The thread of discourse*. Janua Linguarum. Mouton De Gruyter.

Hixon, B. and Passonneau, R. J. (2013). Open dialogue management for relational databases. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1082–1091, Atlanta, Georgia.

Hovy, E. (1993). Automated discourse generation using discourse structure relations. *Artificial Intelligence*, 63:341–385.

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

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

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

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

Isard, A., Oberlander, J., Matheson, C., and Androutsopoulos, I. (2003). Speaking the users' languages. *IEEE Intelligent Systems*, 18(1):40–45.

Johnson, M. (2007). Why doesn't EM find good HMM POS-taggers? In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 296–305, Prague, Czech Republic.

Joshi, A. K. (1987). The relevance of tree adjoining grammar to generation. In Kempen, G., editor, *Natural Language Generation*, volume 135 of *NATO ASI Series*, pages 233–252. Springer Netherlands.

Karamanis, N. (2003). *Entity Coherence for Descriptive Text Structuring*. PhD thesis, University of Edinburgh.

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

Kasper, R. T. (1989). A flexible interface for linking applications to penman's sentence generator. In *Proceedings of the workshop on Speech and Natural Language*, HLT '89, pages 153–158, Philadelphia, Pennsylvania.

Katz, S. M. (1987). Estimation of probabilities from sparse data for the language model component of a speech recognizer. In *IEEE Transactions on Acoustics, Speech and Signal Processing*, pages 400–401.

Kay, M. (1984). Functional unification grammar: a formalism for machine translation. In *Proceedings of the 10th International Conference on Computational Linguistics and 22nd annual meeting on Association for Computational Linguistics*, ACL '84, pages 75–78, Stanford, California.

Kay, M. (1996). Chart generation. In *Proceedings of the 34th Annual Meeting on Association for Computational Linguistics*, ACL '96, pages 200–204, Santa Cruz, California.

Kibble, R. and Power, R. (2004). Optimising referential coherence in text generation. *Computational Linguistics*, 30(4):401–416.

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

Klein, D. and 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*, pages 478–485, Barcelona, Spain.

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

Klein, D. and Manning, C. D. (2002). A generative constituent-context model for improved grammar induction. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, pages 128–135, Philadelphia, Pennsylvania.

Kneser, R. and Ney, H. (1995). Improved backing-off for m-gram language modeling. *Acoustics, Speech, and Signal Processing, IEEE International Conference on*, 1:181–184.

Knight, K. and Hatzivassiloglou, V. (1995). Two-level, many-paths generation. In *Proceedings of Association for Computational Linguistics*, pages 252–260, Cambridge, MA.

Konstas, I. and Lapata, M. (2012a). Concept-to-text generation via discriminative reranking. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 369–378, Jeju Island, Korea.

Konstas, I. and Lapata, M. (2012b). Unsupervised concept-to-text generation with hypergraphs. In *Proceedings of the 2012 Conference of the North American Chapter* *of the Association for Computational Linguistics: Human Language Technologies*, pages 752–761, Montréal, Canada.

Konstas, I. and Lapata, M. (2013a). A global model for concept-to-text generation. *Journal of Artificial Intelligence Research*, 48:305–346.

Konstas, I. and Lapata, M. (2013b). Inducing document plans for concept-to-text generation. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1503–1514, Seattle, Washington, USA.

Lavoie, B. and Rambow, O. (1997). A fast and portable realizer for text generation systems. In *Proceedings of the fifth conference on Applied natural language processing*, ANLC '97, pages 265–268, Washington, DC.

Li, Z. and 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*, pages 40–51, Suntec, Singapore.

Li, Z. and Khudanpur, S. (2009). Forest reranking for machine translation with the perceptron algorithm. In *GALE Book*. GALE.

Liang, P., Bouchard-Côté, A., Klein, D., and 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*, pages 761–768, Sydney, Australia.

Liang, P., Jordan, M., and 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*, pages 91–99, Suntec, Singapore.

Lu, W. and 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*, pages 1611–1622, Edinburgh, Scotland, UK.

Mann, W. C. and Thompson, S. A. (1988). Rhetorical structure theory: Toward a functional theory of text organization. *Text*, 8(3):243–281.

Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. (1993). Building a large annotated corpus of english: the Penn treebank. *Computational Linguistics*, 19(2):313–330.

Marr, D. (1976). Early processing of visual information. *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, 275(942):pp. 483–519.

McDonald, R., Hall, K., and Mann, G. (2010). Distributed training strategies for the structured perceptron. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 456–464, Los Angeles, CA.

McKeown, K. R. (1985a). Discourse strategies for generating natural-language text. *Artificial Intelligence*, 27(1):1–41.

McKeown, K. R. (1985b). *Text generation : using discourse strategies and focus constraints to generate natural language text*. Cambridge University Press.

Mel'čuk, I. (1988). *Dependency Syntax: Theory and Practice*. State University of New York Press.

Mellish, C., Knott, A., Oberlander, J., and O'Donnell, M. (1998). Experiments using stochastic search for text planning. In *Proceedings of International Natural Language Generation*, pages 98–107, New Brunswick, NJ.

Miller, G. A. (1995). Wordnet: A lexical database for english. *Communications of the ACM*, 38:39–41.

Moore, J. D. and Paris, C. L. (1989). Planning text for advisory dialogues. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics*, pages 203–211, Vancouver, British Columbia, Canada.

Moore, R. C. (2002). A complete, efficient sentence-realization algorithm for unification grammar. In *Proceedings of the 2002 International Natural Language Generation Conference*, pages 41–48.

Nederhof, M.-J. and 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*, pages 160–167, Sapporo, Japan.

Papineni, K., Roukos, S., Ward, T., and 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*, pages 311–318, Philadelphia, Pennsylvania.

Portet, F., Reiter, E., Gatt, A., Hunter, J., Sripada, S., Freer, Y., and Sykes, C. (2009). Automatic generation of textual summaries from neonatal intensive care data. *Artificial Intelligence*, 173(7-8):789–816.

Rambow, O. and Korelsky, T. (1992). Applied text generation. In *Proceedings of the Third Conference on Applied Natural Language Processing*, pages 40–47, Trento, Italy.

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. (1994). Has a consensus NL generation architecture appeared, and is it psycholinguistically plausible? In *Proceedings of the Seventh International Workshop on Natural Language Generation*, INLG '94, pages 163–170, Kennebunkport, Maine.

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

Reiter, E., Robertson, R., and Osman, L. M. (2003). Lessons from a failure: Generating tailored smoking cessation letters. *Artificial Intelligence*, 144(12):41 – 58.

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

Scott, D. and de Souza, C. S. (1990). Getting the message across in RST-based text generation. In Dale, R., Mellish, C., and Zock, M., editors, *Current Research in Natural Language Generation*, pages 47–73. Academic Press, New York.

Shen, L., Sarkar, A., and Och, F. J. (2004). Discriminative reranking for machine translation. In *HLT-NAACL 2004: Main Proceedings*, pages 177–184, Boston, Massachusetts.

Shieber, S. M. (1988). A uniform architecture for parsing and generation. In *Proceedings of the 12th Conference on Computational Linguistics - Volume 2*, COLING '88, pages 614–619, Budapest, Hungry.

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

Snyder, B. and Barzilay, R. (2007). Database-text alignment via structured multilabel classification. In *IJCAI'07: Proceedings of the 20th international joint conference on Artifical intelligence*, pages 1713–1718, San Francisco, CA. Morgan Kaufmann Publishers Inc.

Sripada, S. G., Reiter, E., Hunter, J., and 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*, pages 187–196. ACM Press.

Stolcke, A. (2002). SRILM – an extensible language modeling toolkit. In Hansen, J. H. L. and Pellom, B. L., editors, *INTERSPEECH*. ISCA.

Stone, M. and Doran, C. (1997). Sentence planning as description using tree adjoining grammar. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*, pages 198–205, Madrid, Spain.

Stone, M. and Webber, B. L. (1998). Textual economy through close coupling of syntax and semantics. In *Proceedings of the International Workshop on Natural Language Generation*, volume cmp-lg/9806020. Association for Computational Linguistics.

Toutanova, K., Klein, D., Manning, C. D., and 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*, pages 173–180, Edmonton, Canada.

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

White, M. (2004). Reining in ccg chart realization. In *Proceedings of the 2004 International Natural Language Generation Conference*, pages 182–191.

White, M. and Baldridge, J. (2003). Adapting chart realization to CCG. In *Proceedings of 9th European Workshop on Natural Language Generation*, Budapest, Hungary.

Wong, Y. W. and 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*, pages 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. and 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*, pages 678–687, Prague, Czech Republic.