FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

# DOCTORAL THESIS

Ondřej Dušek

# Novel Methods for Natural Language Generation in Spoken Dialogue Systems

Institute of Formal and Applied Linguistics

Supervisor: Ing. Mgr. Filip Jurčíček, Ph.D.

Study Program: Computer Science
Specialization: Computational Linguistics

Prague 2017

I declare that I carried out this doctoral thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

Prague, April 12, 2017 Ondřej Dušek

**Title:** Novel Methods for Natural Language Generation in Spoken Dialogue Systems

**Author:** Ondřej Dušek

**Department:** Institute of Formal and Applied Linguistics

**Supervisor:** Ing. Mgr. Filip Jurčíček, Ph.D., Institute of Formal and Applied Linguistics

**Abstract:**

This thesis explores novel approaches to natural language generation (NLG) in spoken dialogue systems (i.e., generating system responses to be presented the user), aiming at simplifying adaptivity of NLG in three respects: domain portability, language portability, and user-adaptive outputs.

Our generators improve over state-of-the-art in all of them: First, our generators, which are based on statistical methods (A* search with perceptron ranking and sequence-to-sequence recurrent neural network architectures), can be trained on data without fine-grained semantic alignments, thus simplifying the process of retraining the generator for a new domain in comparison to previous approaches. Second, we enhance the neural-network-based generator so that it takes preceding dialogue context into account (i.e., user's way of speaking), thus producing user-adaptive outputs. Third, we evaluate several extensions to the neural-network-based generator designed for producing output in morphologically rich languages, showing improvements in Czech generation.

In addition, we compare different NLG architectures (a traditional two-step pipeline with separate sentence planning and surface realization steps and a joint, end-to-end approach), and we collect and make freely available two novel training datasets for NLG.

**Keywords:** natural language generation, spoken dialogue systems, adaptivity, dialogue entrainment, multilingualism

**Název práce:** Nové metody generování promluv v dialogových systémech

**Autor:** Ondřej Dušek

**Katedra:** Ústav formální a aplikované lingvistiky

**Vedoucí práce:** Ing. Mgr. Filip Jurčíček, Ph.D.,
Ústav formální a aplikované lingvistiky

**Abstrakt:**

Tato disertační zkoumá nové přístupy ke generování přirozeného jazyka (NLG) v hlasových dialogových systémech, tj. generování odpovědí systému pro uživatele. Zaměřuje se přitom na zlepšení adaptivity NLG ve třech ohledech: přenositelnost mezi různými doménami, přenositelnost mezi jazyky a přizpůsobení výstupu uživateli.

Ve všech ohledech dosahují naše generátory zlepšení oproti dřívějším přístupům: 1) Naše generátory, založené na statistických metodách (prohledávání A* s perceptronovým rerankerem a architektuře rekurentních neuronových sítí sequence-to-sequence), lze natrénovat na datech bez podrobného sémantického zarovnání slov na atributy vstupní reprezentace, což dovoluje jednodušší přetrénování pro nové domény než předchozí přístupy. 2) Generátor založený na neuronových sítích dále rozšiřujeme tak, že při generování bere v potaz kontext dosavadního dialogu (tj. i uživatelův způsob vyjadřování) a vytváří tak výstup přizpůsobený uživateli. 3) Vyhodnocujeme také několik úprav systému založeného na neuronových sítích, které jsou zaměřeny na generování výstupu v morfologicky bohatých jazycích, a ukazujeme zlepšení v generování češtiny.

Při našich experimentech navíc porovnáváme různé architektury NLG (tradiční dvojfázové zpracování s odděleným větným plánovačem a povrchovým realizátorem a integrovaný, jednofázový přístup). Pro trénování generátorů jsme též sestavili a zveřejnili dvě nové datové sady.

**Klíčová slova:** generování přirozeného jazyka, dialogové systémy, adaptivita, entrainment v dialogu, vícejazyčnost

## Acknowledgements

*First, I would like to express my thanks to my supervisor Filip Jurčíček for inspiring this thesis, for his guidance and advice, for his continuing attention and support, for our many helpful discussions, and for keeping me focused and motivated.*

*I am also very grateful to all my colleagues and friends at the Institute of Formal and Applied Linguistics for their help, advice, and encouragement. I thank my senior colleagues and mentors, Jan Hajič, Zdeněk Žabokrtský, and others, for inspiring and supporting me. I would also like to thank my fellow Ph.D. (ex-)students, Jindřich Helcl, David Mareček, Michal Novák, Ondřej Plátek, Martin Popel, Rudolf Rosa, Aleš Tamchyna, Miroslav Vodolán, Lukáš Žilka, and others, for lots of interesting debates. Also, many thanks to all the ladies and gentlemen at our Institute who kept my spirits high by sharing beers, songs, stories, and adventures with me. A special thanks goes to Jindřich Libovický for his helpful comments on the draft of this thesis.*

*Thanks to all volunteers who helped evaluate the outputs of my NLG systems.*

*I also want to thank my parents and the whole of my family for their unending support and encouragement.*

*Most of all, I would like to thank my wife Jana for her love, friendship, care, and patience. She has always been there for me, always had my back, and helped me through all the stress and difficulties.*

The work on this thesis was supported by the Charles University Grant Agency (grant 2058214), by the Ministry of Education, Youth and Sports of the Czech Republic (project LK11221), and by the EU 7th Framework Programme grant QTLeap (No. 610516). It used resources stored and distributed by the LINDAT/CLARIN project of the Ministry of Education, Youth and Sports of the Czech Republic (projects LM2010013 and LM2015071).

# Contents

# 1 Introduction

Natural language generation (NLG), a conversion of an abstract and formalized representation of a piece of information into a natural language utterance, is an integral part of various natural language processing (NLP) applications. It is used in the generation of short data summaries, question answering, machine translation (MT), and also in spoken dialogue systems (SDSs), the latter area being the focus of the present thesis.

SDSs are computer interfaces allowing users to perform various tasks or request information using spoken dialogue. They are typically designed to provide information about a specified domain, such as air travel (Walker et al., 2001b), restaurants (Rieser et al., 2010; Young et al., 2013), or public transport

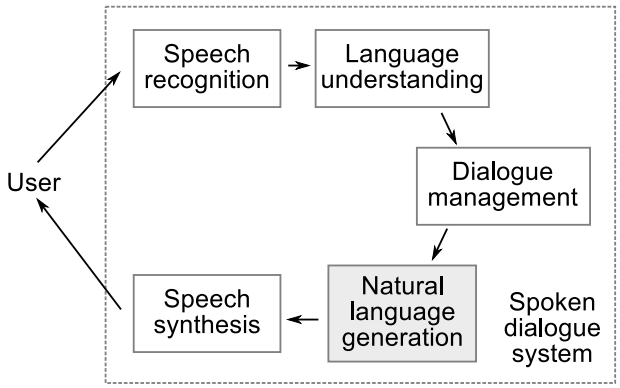


Figure 1.1: A typical SDS pipeline, with the NLG component highlighted

hello()
*Hello, this is dialogue system X. How can I help?*

inform(name="Baker's Arms", venue=restaurant, foodtype=English,
pricerange=moderate)
*The restaurant Baker's Arms serves English food in the moderate price range.*

request(departure_time)
*What time do you wish to leave?*

inform_no_match(vehicle=bus, departure_time=11:00pm)
*I am sorry but I cannot find a bus connection at 11:00pm.*

Figure 1.2: Examples of the DA meaning representation, along with natural language paraphrases (for restaurant and public transport domains)

(Raux et al., 2005; Dušek et al., 2014).[1] A typical SDS pipeline (Rudnicky et al., 1999; Raux et al., 2003; Young et al., 2013; Jurčíček et al., 2014; see Figure 1.1) starts with speech recognition and language understanding modules, which deliver the semantic content of user utterances to the dialogue manager, the central component responsible for the behavior of the system. The task of NLG is then to convert an abstract representation of the system response coming from the dialogue manager into a natural language sentence, which is passed on to a text-to-speech synthesis module. NLG is thus responsible for accurate, comprehensible, and natural presentation of information provided by the SDS and has a significant impact on the overall perception of the system by the user.

To represent both user and system actions, task-oriented SDSs typically use a domain-specific shallow meaning representation (MR) such as dialogue acts (DAs; Young et al., 2010), consisting of a dialogue act type or dialogue action, roughly corresponding to speech acts of Austin and Searle (Korta and Perry, 2015), e.g., *inform*, *request*, or *hello*, and an optional set of attributes (slots) and their values (see Figure 1.2).[2] DAs are thus the input to a NLG component in a SDS, and they correspond to a natural language sentence or a small number of sentences on the output.

[1] The voice assistants such as Google Now/Google Home, Apple Siri, Microsoft Cortana, or Amazon Alexa, which have gained a lot of attention and popularity recently, are examples of advanced SDSs supporting multiple domains (task scheduling, home automation, news, etc.).

[2] While DAs are originally based on speech acts and pragmatics theory (cf. Walker and Passonneau, 2001), their form used in this thesis and most current SDSs is mainly concerned with semantics, not pragmatics of the utterances.

In this introductory chapter, we will first explain our motivation for research in NLG for SDSs in Section 1.1, then list the main objectives and contributions of the present thesis in Section 1.2. Section 1.3 introduces the contents of the following chapters, and Section 1.4 lists machine learning methods and algorithms which are used or referred to but not explained in this thesis, providing some pointers to basic literature.

## 1.1 Motivation

The main motivation for this work has been the relative lack of statistical approaches in NLG for SDSs that are practically usable. While the usage of statistical methods and trainable modules is not new in NLG, their adoption mostly remained limited in spoken dialogue systems, until very recently. Traditionally, NLG systems were built as pipelines of mostly handcrafted modules. In SDSs, the NLG component has often been reduced to a simple template-filling approach (Rudnicky et al., 1999; Jurčíček et al., 2014).[3] Although statistical approaches in NLG have advanced greatly during the past year with the advent of neural network (NN) based systems (see Section 2.3), they still leave room for improvement in terms of naturalness, adaptability, and linguistic insight. Present NN-based NLG (Wen et al., 2015b,a; see Section 2.3) has only been evaluated on relatively large English datasets and lacks the ability to adapt to a particular user.

## 1.2 Objectives and Contributions

The main aim of the present thesis is to explore the usage of statistical methods in NLG for SDSs and advance the state-of-the art among the dimensions outlined in the previous section – naturalness and adaptability. First, we focus on enabling fast reuse in new domains and languages and second, we aim at adapting the structure and lexical choice in generated sentences to the communication goal, to the current situation in the dialogue, and to the particular user (e.g., by aligning vocabulary to the expressions uttered by the user). This work thus not only brings a radical improvement over NLG systems based on handwritten rules or domain-specific templates, which have been the norm in the field until very recently, but also represents an important contribution to re-

[3] This also applies to other areas where NLG is used, e.g., in personalized web sites such as Facebook or LinkedIn.

cent works in statistical NLG by experimenting with deep-syntactic generation, multilingual NLG, and user-adaptive models.

Our experiments, and also the main contributions of this thesis, proceed along the following key objectives:

**A) Generator easily adaptable for different domains.** We create a generator that can be fully and easily retrained from data for a different domain. Unlike previous methods, our generator does not require fine-grained alignments between elements of the input meaning representation and output words and phrases, and learns from unaligned pairs of input DA and output sentences. We will show two different novel approaches to NLG trainable from unaligned data.

**B) Generator easily adaptable for different languages.** Here, we explore the adaptation of a rule-based general-domain surface realizer to a new language, simplify it by introducing statistical components, and show that porting to a different language does not require excessive efforts. In addition, we experiment with fully statistical NN-based NLG on both English and Czech for the first time.

**C) Generator that adapts to the user.** We create a first fully trainable context-aware NLG system that is able to adapt the generated responses to the form of the user's requests, thus creating a natural level of linguistic alignment in the dialogue.

**D) Comparing different NLG system architectures.** We experiment with both major approaches used in modern NLG systems – pipeline (separating high-level sentence structuring from surface grammatical rules) and joint – and compare their results on the same dataset.

**E) Dataset availability for NLG in SDSs.** We address the limited availability of datasets for NLG in task-oriented SDSs by collecting and publicly releasing two different novel datasets: the first dataset for training context-aware NLG systems and the first Czech NLG dataset (which is one of very few non-English sets).

## 1.3 Chapter Guide

The remainder of this thesis is structured and addresses the main objectives specified in Section 1.2 in the following manner:

The immediately following two chapters are dedicated to rather theoretical questions, providing background for all objectives, especially Objective D (comparing different approaches). Chapter 2 represents an overview over current state-of-the-art in NLG for SDSs, focusing on adaptive and trainable methods and comparing different approaches and implementations. Notes on available datasets and evaluation methods are also included. Chapter 3 then provides some general background and preliminary considerations with respect to our own approach to NLG, describing the data formats and methods that we use in the rest of this thesis.

The remaining chapters but for the last one are an account of our experiments introducing novel methods to NLG for SDSs to improve along the objectives set in Section 1.2. Chapters 4 and 5 describe our experiments with non-neural NLG, divided into sentence planning and surface realization stages. Note that we proceed in the order in which these stages needed to be implemented, which is inverse to the order of their application in the actual NLG system: We first establish a way of converting the intermediate sentence plan representation into natural language strings in Chapter 4, then experiment with converting DAs into sentence plans in Chapter 5. The realization experiments in Chapter 4 mainly address language and domain portability (Objectives B and A). The sentence planning experiments in Chapter 5 concentrate on easy domain portability only (Objective A).

Chapters 6, 7 and 8 present our three different experiments with applying recurrent neural networks (RNNs) in NLG. First, Chapter 6 introduces the basics of our neural NLG approach, shows an improvement over non-neural results from Chapter 5 and compares two different NLG system architectures (two-step pipeline and joint, direct generation) using the same RNN. We use the surface realizer created in Chapter 4 for the pipeline approach. Chapter 6 thus addresses Objectives A (easy domain portability, extending on Chapter 5) and D (comparing different NLG approaches). Second, Chapter 7 extends the RNN model from Chapter 6 to take the preceding user utterance into account and generate outputs appropriate for the current dialogue context. Here we address Objective C (adapting to the user). To test our model extensions, we collect a novel context-aware dataset and release it publicly, thus addressing Objective E. Third and finally, Chapter 8 deals with applying and extending our RNN model from Chapter 6 to a different language, Czech. We address issues not previously encountered in English, mainly connected to rich Czech morphology. To evaluate our models, we collect the first Czech NLG dataset, which is now also publicly available. Chapter 8 thus addresses Objectives B and E (language portability and dataset availability).

In the final Chapter 9, we provide a final account of all our experiments and include a few concluding remarks and possible future work ideas.

## 1.4 Machine Learning Essentials

While this thesis aims to be as self-contained as possible, it does assume a certain level of knowledge in NLP and machine learning on the part of the reader. We provide here a list of standard NLP concepts and machine learning techniques that are used without explanation later on, along with very brief, intuitive descriptions and references to basic literature:

**$n$-gram** is simply an $n$-tuple of consecutive tokens in a sequence (Manning and Schütze, 2000, p. 191ff.). $n$-grams of lower orders are called unigrams, bigrams, and trigrams for $n = 1, 2, 3$, respectively.

**$n$-gram Language Model** (LM) is a Markov model of the $n$-1-th order that predicts the a probability distribution over the next token in the sentence based on the preceding $n-1$ tokens (Manning and Schütze, 2000, p. 191ff.; Koehn, 2010, p. 181ff.). The probabilities are typically estimated from corpora, and various smoothing techniques are used to mitigate adverse effects of data sparsity (e.g., Kneser and Ney, 1995; Koehn, 2010, p. 188ff.).

**Perceptron** (Bishop, 2006, p. 192ff.) is, in its basic form, a binary classification supervised learning algorithm. It assumes a model of the form

$$y = f(\mathbf{w} \cdot \mathbf{x}) \tag{1.1}$$

In (1.1), **x** represents features of an object, **w** the corresponding weights, $y \in \{-1, +1\}$ is the object class, and $f$ is the step function:

$$f(z) = \begin{cases} +1 \text{ if } z \geq 0 \\ -1 \text{ if } z < 0 \end{cases} \tag{1.2}$$

The perceptron uses the following algorithm to learn the weights **w**:

1. Classify an instance **x** using current feature weights **x**:

$$\hat{y} := f(\mathbf{w} \cdot \mathbf{x}) \tag{1.3}$$

2. In case of a classification error ($\hat{y} \neq y$, where $y$ is the true class), update the weights:

$$\mathbf{w} := \mathbf{w} + \alpha \cdot (y - \hat{y}) \cdot \mathbf{x} \tag{1.4}$$

**Logistic Regression** (Bishop, 2006, p. 205ff.) is a discriminative model for binary classification very similar to the perceptron, in the following form:

$$y = \sigma(\mathbf{w} \cdot \mathbf{x}) \tag{1.5}$$

In (1.5), $\mathbf{w}, \mathbf{x}$ are the same as in (1.1), $y \in \{0, 1\}$ is the object class, and $\sigma$ is the logistic function:

$$\sigma(z) = \frac{1}{1 + \exp(-z)} \tag{1.6}$$

The prediction is an estimate of the probability that $y = 1$. The model is usually fitted using maximum likelihood estimation (Manning and Schütze 2000, p. 197ff.; Bishop 2006, p. 23).

**Conditional Random Fields** (CRFs; Lafferty et al., 2001; Sutton and McCallum, 2012) are discriminative models for structured data, mostly applied to sequences (linear chain CRFs). A linear chain CRF predicts a sequence of classes $\mathbf{y}$ belonging to an input sequence of objects $\mathbf{x}$ by modeling the conditional probability $P(\mathbf{y}|\mathbf{x})$:

$$P(y_t|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\big(\sum_{k=1}^{K} w_k f_k(y_t, y_{t-1}, \mathbf{x})\big) \tag{1.7}$$

In (1.7), the probability of an item $y_t$ in the sequence of classes depends on the previous class $y_{t-1}$ and the whole input sequence of objects $\mathbf{x}$ through a series of arbitrary feature functions $f_k$ and their corresponding weights $w_k$. $Z$ stands for a normalization constant.

**Neural Network** (NN) models (Bishop, 2006, p. 225ff.; Goodfellow et al., 2016, p. 168ff.) are in essence an extension of the perceptron/logistic regression approach, using multiple interconnected basic units. A basic NN unit (neuron) typically consists of a dot product of inputs $\mathbf{x}$ and weights $\mathbf{w}$, with an optional non-linear transformation $g$ applied afterwards:

$$o = g(\mathbf{w} \cdot \mathbf{x}) \tag{1.8}$$

Typical choices of $g$ include the logistic (sigmoid) function $\sigma$ (1.6), the hyperbolic tangent function tanh (1.9), and the softmax function (1.10).

$$\tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} = 2\sigma(2x) - 1 \tag{1.9}$$

$$\text{softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{|\mathbf{x}|} e^{x_j}} \tag{1.10}$$

The output of the neuron $o$ can be fed to other connected neurons. The whole NN thus builds an acyclic graph of neurons, which is typically divided into layers (feedforward networks). As a rule, NNs are trained using gradient-based methods (see e.g., Goodfellow et al., 2016, 151ff.).

**Recurrent Neural Networks** (RNNs; Goodfellow et al., 2016, p. 373ff.) represent a special type of NNs where the same group of neurons (called a cell) with identical weights is repeatedly applied to elements of a sequence, such as tokens of a sentence. The inputs of a cell include a representation of the current element as well as the outputs of the preceding cell. Thanks to their recurring architecture, RNNs can be applied to variable-length input sequences.

**Neural Language Models** (RNN LMs; Mikolov et al., 2010, 2011) are LMs based on an RNN. While an $n$-gram LM predicts the probability of a next token based just on simple corpus statistics over the immediately preceding $n - 1$ words in the sentence, an RNN LM trains its RNN cells to predict next token probabilities based on all previous tokens in the sentence, thus allowing to model long-distance dependencies. Furthermore, the network can be initialized specially to condition the model on an external input (see Sections 2.3 and 6.2).

Same as a $n$-gram LM, an RNN LM allows generating sentences directly from the model (Graves, 2013), using greedy generation (in each step, choose the most probable token in the dictionary), sampling according to the probability distribution over possible next tokens, or beam search (Bengio et al., 2015; Cho, 2016).

# 2

# State of the Art: Adaptive Methods in NLG

In this chapter, we give a brief introduction into the problem of NLG, focusing on its application in spoken dialogue systems, we list state-of-the-art trainable and adaptive approaches implemented for various NLG system components and briefly discuss available training data.

First, we give a general textbook definition of the problem of NLG in Section 2.1, along with the description of the basic stages into which the ideal, textbook NLG pipeline is divided. We then provide some remarks as to the practical implementation of these stages, and list the main advantages and disadvantages of handcrafted and trainable NLG systems, the latter of which are our main concern for the remainder of the chapter (and the whole thesis).

The following two sections give a detailed state-of-the-art overview over various trainable/adaptive approaches to NLG.[1] Section 2.2 is dedicated to the individual approaches to introducing adaptivity into all stages of traditional pipeline NLG systems and focuses primarily on generators used in dialogue systems. Note that similar methods are used at different stages and thus the order in which they are described is not necessarily chronological. Section 2.3 then describes attempts at an integrated approach to making the whole NLG pipeline adaptive, first listing pre-NN approaches, then finishing with most recent RNN-based models.

The final Section 2.4 then focuses on the necessary prerequisite to any trainable system: training datasets for NLG. We show that unlike in other NLP areas,

[1] The overview only includes works published up to June 2016 (with a few exceptions).

publicly NLG datasets (especially those oriented on SDSs) have been rather rare.

## 2.1 The Varied Landscape of NLG Systems

In general, natural language generation is defined as the task of presenting information according to a pre-specified communication goal and in a natural language understandable to human users (Dale et al., 1998). Given input data (in any format) and a communication goal (e.g., to describe the data or receive user reaction), the system should produce a natural language string that is relevant, well-formed, grammatically correct, and fluent.

The standard "textbook" description of a NLG system (Reiter and Dale, 2000) involves a pipeline consisting of three main phases:

1. *Content planning* (also referred to as *content selection* or *document planning*). The system selects relevant content from the input data and performs basic structuring of this content. The output of this phase is a *content plan*, usually a structured listing of the content to be presented.

2. *Sentence planning* (also called *microplanning*) – a detailed sentence shaping and expression selection. The output of this phase is a *sentence plan*, usually a detailed syntactic or semantic representation of the output sentence(s).

3. *Surface realization* is in essence a linearization of the sentence plan according to the grammar of the target language; it includes word order selection and morphological inflection. The output of this phase is natural language text.

The content selection phase is said to decide on "what to say", while surface realization determines "how to say it". The sentence planning phase is concerned in part with both tasks, serving as an interface between them (Meteer, 1990, cited by Dale et al., 1998). While the input and intermediate formats vary greatly in different systems and different usage areas, the general approaches and algorithms are often transferable. Therefore, we also include NLG systems that are focused on other usage areas than SDS into the following description.

### Partial Implementations of the Pipeline

Most NLG systems follow the standard pipeline more or less closely, but only a few of them implement it as a whole. Many generators focus only on one of the

phases while using a very basic implementation of the other or leaving it out completely.

Systems concerned with human-readable data presentation for a specific domain tend to implement all stages (e.g., weather reports in Reiter et al., 2005); domain-independent generators tend to focus on the latter stages and typically require a detailed content plan (Walker et al., 2001a) or even semantic description according to a grammar (Ptáček and Žabokrtský, 2007; Belz et al., 2011) as their input. Some generators are even only concerned with finding the best word order for a given bag of words (Gali and Venkatapathy, 2009; Zhang and Clark, 2011).

In many SDSs, content planning is handled by the dialogue manager and the NLG component only performs sentence planning and surface realization. On the other hand, NLG systems in SDSs (Rambow et al., 2001) often focus just on the content planning or sentence planning stage and include sophisticated methods of selecting the best way of presenting requested information to the user (e.g., Walker et al., 2001a; Moore et al., 2004) while using a simple implementation or reusing an off-the-shelf system for the final generation stage.

## Pipeline and Joint Approaches

The whole structure of individual NLG systems also varies, depends on the particular area of usage, and is often arbitrary. While some systems keep the traditional division of the tasks along a pipeline, others opt for a join approach. Both architectures can offer their own advantages.

Dividing the problem of NLG into several subtasks makes the individual subtasks simpler. For instance, a sentence planner can abstract away from complex surface syntax and morphology and only concern itself with a high-level sentence structure. It is also possible to reuse third-party modules for parts of the generation pipeline (Walker et al., 2001a). For surface realization, developing a handcrafted reusable, domain-independent module with a reasonable coverage is not too difficult, as we show in Chapter 4.

On the other hand, the problem of pipeline approaches in general is error propagation. Downstream modules either copy the errors from the input or they need to handle them specially; this is not needed in joint, one-step approaches. In addition, joint methods do not need to model intermediate structures explicitly (Konstas and Lapata, 2013). Therefore, no training sentence plans or content plans are required for statistical joint systems. However, pipeline approaches can satisfy the need for explicit intermediate structures

in training data by using existing automatic analysis tools to obtain them (see Section 3.4).

## Handcrafted and Trainable Methods

Traditional NLG systems are based on procedural rules (Bangalore and Rambow, 2000; Belz, 2005; Ptáček and Žabokrtský, 2007), template filling (Rudnicky et al., 1999; van Deemter et al., 2005), or grammars in various formalisms, such as functional unification grammar (Elhadad and Robin, 1996) or combinatory categorial grammar (CCG) (White and Baldridge, 2003). Such rule-based generators are still used frequently today. Their main advantage is implementation simplicity; they are very fast and can be adjusted in a straightforward way, allowing for a fine-grained customization to the domain at hand and a direct control over the fluency of the output text.[2] Moreover, large-coverage rule-based general-domain surface realizers (e.g., Elhadad and Robin, 1996; Lavoie and Rambow, 1997) can be reused in new systems generating into the same output language.

On the other hand, many rule-based systems struggle to achieve high coverage in larger domains (White et al., 2007) and are not easy to adapt for different domains and/or languages. Multilingual rule-based generation systems (Bateman, 1997; Allman et al., 2012; Dannélls, 2012) typically use a shared semantic representation but require handwritten grammar rules for each new language they support. Rule-based systems also tend to exhibit little variation in the output, which makes them appear repetitive and unnatural.

Various approaches have been taken to make NLG output more flexible and natural as well as to simplify its reuse in new domains. While statistical methods and trainable modules in NLG are not new (cf. Langkilde and Knight, 1998), their adoption has been slower than in most other subfields of NLP, such as speech recognition, syntactic parsing, or MT. Several different research paths were pursued for statistical NLG in the last decade; many of them focus on just one of the generation stages or on enhancing the capabilities of an existing rule-based generator, e.g., by introducing parameters that lead to more output diversity. Only during the past year or two, fully trainable NN-based generators (e.g., Wen et al., 2015b,a, but also work developed in the course of the present thesis) have been dominating the field.

[2] See also Belz and Kow (2010a)'s research comparing the fluency of rule-based and statistical NLG systems.

## 2.2 Introducing Adaptive Components into Pipeline NLG

The rule-based pipeline NLG systems of the 1990s evolved quickly to include statistical components at least in some parts of the pipeline. In this section, we list the most notable approaches, divided according to the respective pipeline stages, focusing mostly on NLG systems for SDSs.

### Adaptive Content Planning

Content planning within SDSs is closely related to dialogue management and the NLG approaches presented in this thesis do not include this step. However, the algorithms applied in adaptive content planning for SDSs are relevant for our work as they include user-adaptive techniques and can be transferred to the later stages of the pipeline.

First attempts at introducing adaptivity into content plans for SDSs were targeted at custom-tailoring information presentation for the user and involved a parametric user model (Moore et al., 2004; Walker et al., 2004; Carenini and Moore, 2006). They used a handcrafted content planner that allowed the user to specify their preferences regarding the output by answering a set of simple questions (ranking certain attributes of the output by their importance). The user's answers were then transformed to parameter weights for the planner using simple heuristic functions. While such systems bring user adaptivity and variation, they require the content planner to be not only handcrafted for the domain at hand but also flexible regarding the parameter settings.

A more recent line of research in content planning for SDSs (Lemon, 2008; Rieser and Lemon, 2010; Lemon et al., 2012) recasts the problem as planning under uncertainty and employs reinforcement learning (Sutton and Barto, 1998) to find the optimal presentation strategy for the content requested by the user. In this setting, content planning is modeled as a Markov decision process in a space of possible generation states connected by lower-level, single-utterance NLG actions, such as "summarize search results" or "recommend the best item". The generator plans a sequence of these actions to achieve the communication goal, i.e., having the user choose one of the results presented in as few lower-level actions as possible. Achieving the goal represents a reward in the reinforcement learning setting, while the system is penalized for the amount of actions taken. The system uses the SHARSHA reinforcement learning algorithm (Shapiro and Langley, 2002) to learn the best policy of state-action transitions by repeatedly

generating under the current policy and updating the value estimates for state-action pairs. A user simulator based on $n$-grams[3] of user and system actions (Eckert et al., 1997) replaces humans in the training loop, allowing for a large number of iterations.

## Adaptive Sentence Planning

First trainable sentence planners took the overgeneration and ranking approach originally introduced in surface realization (see below), as in the SPoT system (Walker et al., 2001a, 2002) and its extension, SPaRKy (Stent et al., 2004): More variants of the output are randomly generated and a statistical reranker selects the best variant afterwards. In the SPoT and SPaRKy systems, this involved a rule-based sentence plan generator producing many different sentence plans by using various clause-combining operations over simple statements on the input (e.g., coordination, contrast, or joining through a relative clause or a *with*-phrase). The best sentence plan was subsequently selected by a RankBoost reranker trained on hand-annotated sentence plans. Such systems are adaptive and provide variation in the output, but require a handcrafted base module and are rather computationally expensive.

Variance in the output can be achieved without the high computational cost of overgeneration using a parameter optimization approach. Sentence planners with parameter optimization require a handcrafted base module with a set of parameters whose values are adjusted to produce output with desired properties. Mairesse and Walker (2007) experiment in the PERSONAGE system with linguistically motivated parameters for content and sentence planning to generate outputs corresponding to extroverted and introverted speakers; their system is adaptable but all parameters must be controlled manually. Mairesse and Walker (2008, 2011) further expand the system, employing various machine learning methods to find generator parameters corresponding to high or low levels of the Big Five personality traits (extroversion, emotional stability, agreeableness, conscientiousness, openness to experience). Their classifiers predict the individual generator parameters given the personality settings; they are trained on corpus of generator outputs created under various parameter settings and annotated with personality traits in a crowdsourcing scenario.

Other approaches to adaptive sentence planning focus on entrainment (alignment) of the individual parties in a dialogue, i.e., adapting the outputs to previous user utterances and potentially reusing wording or sentence structure. This is expected to improve the perceived naturalness of the output (Nenkova

[3] $n$-tuples of consecutive actions.

et al., 2008). Current systems exploiting dialogue alignment (Buschmeier et al., 2009; Lopes et al., 2013, 2015) are limited to handwritten rules (see Chapter 7, where this problem is addressed in detail).

## Adaptive Surface Realization

Adaptive, trainable, or statistical surface realizers are not necessarily needed for adaptive NLG as there are large-coverage reusable off-the-shelf realizers available. As noted in Section 2.1, they are often used by NLG systems that experiment with trainable content or sentence planning. Notable examples include the FUF/SURGE realizer (Elhadad and Robin, 1996) based on a unification grammar, which is used, e.g., by Carenini and Moore (2006). Further, the RealPro realizer (Lavoie and Rambow, 1997) generates texts from deep syntactic structures based on the Meaning-Text Theory (Melčuk, 1988), which are produced, e.g., by the sentence planners of Walker et al. (2001a), Stent et al. (2004), and Mairesse and Walker (2007). Another example, White and Baldridge (2003)'s OpenCCG realizer from CCG structures, has been extended with statistical modules (White et al., 2007; White and Rajkumar, 2009) and used by Rajkumar et al. (2011) or Berg et al. (2013).

As mentioned above, first adaptive surface realizers (and the first approaches to adaptive NLG in general) were based on the overgeneration and ranking approach. Here, a grammar-based or a rule-based realizer generates more variants of the output text, which are subsequently reranked according to a separate statistical model. The first generators using this approach employed $n$-gram LMs (Langkilde and Knight, 1998; Langkilde, 2000; Langkilde-Geary, 2002) or tree models (Bangalore and Rambow, 2000) for ranking. Various other reranking criteria were introduced later, including expected text-to-speech output quality (Nakatsu and White, 2006), desired personality traits and expression alignment with the dialogue counterpart (Isard et al., 2006), or a score according to a perceptron classifier trained to match reference sentences using a rich feature set including $n$-gram model scores and syntactic traits (White and Rajkumar, 2009). Same as in sentence planning, the reranking approach achieves greater variance, but has a higher computational cost and still requires a base handcrafted module.

First fully statistical surface realizers were built by automatically inducing grammar rules from a treebank and applying methods based on inverted chart parsing (Kay, 1996). Nakanishi et al. (2005) use a conversion of the Penn Treebank (Marcus et al., 1993) to the head-driven phrase structure grammar (HPSG, Pollard and Sag, 1994) as their realization input; Cahill and van Genabith (2006)

attempt to regenerate the same treebank from a conversion to lexical functional grammar structures (Bresnan, 2001).

The fully trainable surface realizer of Bohnet et al. (2010) is based on the Meaning-Text Theory; they convert treebanks of four languages (English, Spanish, German, and Chinese) (Hajič et al., 2009) into their graph-based deep-syntactic representation. The realizer is a three-step pipeline: they use beam search to decode dependency trees from deep-syntactic trees by starting from an empty tree and attempting to add nodes one-by-one, scoring the results with a support vector machine (SVM)-based ranker (Cristianini and Shawe-Taylor, 2000) along the way. The dependency trees are then linearized using another beam search decoder and SVM scorer, building ordered subsets of nodes from left to right. In the final step, they generate morphological inflection, predicting rules for changing base word forms (lemmas) into inflected forms using a third SVM classifier.

Bohnet et al. (2011b) extend the system to generate from a more abstract representation which better reflects the Meaning-Text Theory semantic layer and does not include auxiliary words such as prepositions or articles. Since there is no longer a one-to-one node correspondence between the source semantic structure and the target dependency trees, they extract tree transducer rules from a treebank and use an SVM to select which rules need to be applied.

The realizer of Ballesteros et al. (2014) further extends Bohnet et al. (2011b)'s system, focusing on generating surface syntactic trees from deep syntax. Instead of using tree transducers, they opt for a fully statistical pipeline of SVM classifiers that first select a part-of-speech auxiliary word pattern for each deep syntax tree node; the auxiliary resulting words are subsequently lexicalized one-by-one. Next, surface syntactic dependencies are resolved between the newly added auxiliaries and the original node. The last step of the pipeline resolves dependencies among the original deep tree nodes.

Several partially or fully statistical realizers from dependency-based structures have been built in connection with the 2011 Generation Challenge surface realization shared task (Belz et al., 2011). Bohnet et al. (2011a) simply adapt Bohnet et al. (2011b)'s system to the different input data format. Rajkumar et al. (2011) take a two-step approach: they first adapt the deep syntactic trees to the CCG formalism by applying a maximum entropy classifier to infer semantic relations missing on the input, then apply the OpenCCG realizer. Stent (2011) and Guo et al. (2011) both approach shallow generation (syntactic tree linearization and word inflection) in a similar fashion to Bohnet et al. (2010) but simpler: They use a combination of tree models and $n$-gram models learned from the

input corpus to linearize the input structures and apply a simple morphological dictionary for inflection.

The fully statistical surface realizers described above focus only on the surface realization step and do not include a sentence planner. They typically attempt to regenerate existing syntactically and semantically annotated corpora (such as Marcus et al., 1993; Hajič et al., 2009) and are tested in a standalone setting. Nevertheless, the division between surface realizers and one-step full NLG (see Section 2.3) is not sharp as various degrees of abstractness are used in the input formalisms. In addition, techniques used in the standalone surface realizers are often applicable to NLG in SDSs.

## 2.3 Joint Approaches to Adaptive NLG

In the recent years, there have been several attempts to address adaptive NLG in an integrated, end-to-end fashion, thus reducing the number of consecutive stages. Most such systems integrate sentence planning and surface realization into a single module and expect data relating to one simple utterance as their input. Many approaches pursued here are a parallel of surface realization techniques (see Section 2.2) since they only differ in the abstraction level of their inputs.

At the time when work on this thesis started and up until recently, the area of trainable end-to-end generation has been rather limited; in practice, a simple template-filling approach was the typical "joint" approach to NLG in SDSs, albeit non-adaptive. Only recently, NN-based end-to-end approaches appeared, including our own work (cf. also Chapters 6, 7, and 8).

In the following, we first list the approaches to generation that do not use neural networks, then continue with a description of recent NN-based NLG systems that emerged in parallel with our own experiments.

### Non-neural

Similarly to first trainable surface realizers, the first trainable joint approaches to NLG for SDSs used a combination of handcrafted and statistical components. Ratnaparkhi (2000) experiments with purely statistical components in a limited setting; he examines a word-by-word beam search approach to phrase generation with a maximum entropy model in a left-to-right $n$-gram-based and a dependency tree based setting. In a follow-up work, Ratnaparkhi (2002) then intersects the dependency-based model with a handcrafted dependency grammar for usage in a SDS. Galley et al. (2001) use a context-free grammar (CFG) (Man-

ning and Schütze, 2000, p. 97ff.) encoded in a finite state transducer (Manning and Schütze, 2000, p. 367ff.) and apply a beam search with $n$-gram-based scores. Oh and Rudnicky (2000) on the other hand take the overgeneration and ranking approach (see Section 2.2), using a handcrafted component to postprocess the outputs of a statistical one. They generate randomly from an $n$-gram language model for a given utterance class (e.g., *inform_flight*, *inform_price*) and select the best output based on heuristic criteria.

Other hybrid approaches to joint NLG used the parametric handcrafted generator approach that has also been applied to sentence planning (see Section 2.2). Paiva and Evans (2005) employ correlation analysis on a text corpus generated under many different settings of their handcrafted generator to find the influence of the individual parameter values on the presence of desired linguistic features in the output. Belz (2008) combines in several different ways a semi-automatically created, ambiguous CFG with rule probabilities and an $n$-gram model estimated from data.

Some of the more recent joint NLG systems do not require a handcrafted base module and can be fully trained from data. They mostly employ techniques similar to those used in statistical MT systems or syntactic parsers, translating from a formal language of semantic description to a natural language, and they are typically tested in a standalone generation scenario (i.e., not as a part of an SDS). Wong and Mooney (2007) experiment directly with a phrase-based MT system of Koehn et al. (2003), comparing and combining it with an inverted semantic parser based on synchronous CFGs. Lu et al. (2009) use tree CRFs over hybrid trees that may include natural language phrases as well as formal semantic expressions. The recent generator of Flanigan et al. (2016) uses the Abstract Meaning Representation (AMR) formalism (Banarescu et al., 2013) as its input and employs techniques similar to phrase-based MT (Dyer et al., 2010): It selects a spanning tree of the input AMR graph, then applies a tree-to-string transducer learned from a corpus.

Other fully trainable generators exploit the simple, flat structure of input databases for many domains, such as weather information, and operate in a phrase-based fashion. Most of them include basic content selection along with the remaining NLG phases. Angeli et al. (2010) generate text from database records through a sequence of classifiers, gradually selecting database records to mention, their specific fields, and the corresponding textual realizations to describe them. Konstas and Lapata (2013) recast the whole problem of NLG as generation from a probabilistic CFG estimated from database records and their descriptions: They search for the best CFG derivation over the input database records and fields and intersect the CFG model with $n$-gram model scores.

Few fully trainable non-neural joint approaches to generation have been applied in the area of SDSs. Mairesse et al. (2010) represent DAs as "semantic stacks", which correspond to natural language phrases and contain DA types, slots (attributes), and their specific values on top of each other. Their generation model uses two dynamic Bayesian networks: the first one performs an ordering of the input semantic stacks, inserting intermediary stacks which correspond to grammatical phrases, the second one then produces a surface realization, assigning concrete words or phrases to the ordered stacks. Dethlefs et al. (2013) approach generation from DAs as a sequence labeling task and use a CRF classifier, assigning a word or a phrase to every one of the ordered triples of DA types, slots, and values on the input. The recent work of Manishina et al. (2016) combines $n$-gram and CRF "translation" models (probabilities of realizations given input concepts) with a fluency $n$-gram model and a concept reordering model in a finite-state transducer framework. Following up, i.a., on our work described in Chapter 5, Lampouras and Vlachos (2016) in their recent experiments apply imitation learning to directly optimize for word-overlap based evaluation metrics. Similarly to Angeli et al. (2010) and Konstas and Lapata (2013) (see above), they recast generation as a sequence of local decisions, selecting in turn attributes to realize and the corresponding wording.

Most of the generators described above were only tested on very small domains for English and do not include any user-adaptive components. Moreover, these approaches typically assume the alignment of input meaning representation elements to output words as a separate preprocessing step (Wong and Mooney, 2007; Angeli et al., 2010), or require pre-aligned training data (Mairesse et al., 2010; Dethlefs et al., 2013). In addition, their basic algorithm often exploits the properties of a specific input MR shape or formalism, e.g., syntactic trees (Wong and Mooney, 2007; Lu et al., 2009) or flat databases (Angeli et al., 2010; Konstas and Lapata, 2013; Mairesse et al., 2010).

## Approaches Using Neural Networks

In the past year, there have been several new works in end-to-end NLG using conditioned RNN LMs, including our own experiments (see Chapter 6). The new systems bring in a simpler and more powerful architecture: They are typically constructed as an end-to-end solution where the RNN LM conditioned on the input MR generates the output sentence directly.

RNN LMs have been used in various tasks in the field of NLP. First, they replaced $n$-gram LMs in their usual applications, including speech recognition (Mikolov et al., 2010) or MT (Vaswani et al., 2013; Devlin et al., 2014). Recently,

RNN LMs have been applied to various NLP tasks as standalone generators using the sequence-to-sequence (seq2seq) approach, where an encoder NN is applied to encode the input into a fixed-size vector, which is then used to condition the generation from the decoder RNN LM (see Chapter 6 for details). This technique has been first used in MT (Sutskever et al., 2014; Cho et al., 2014; Bahdanau et al., 2015) and image captioning (Vinyals et al., 2015b), later also in syntactic parsing (Vinyals et al., 2015a), poetry generation (Zhang and Lapata, 2014), and morphological inflection generation (Faruqui et al., 2016; Kann and Schütze, 2016, cf. Section 4.4).

Mei et al. (2016) present the first seq2seq-based system for textual NLG known to us; in addition to the basic setup, they dedicate a special part of the network between the encoder and the generation RNN to a two-step content selection. The generation RNN LM has access to the content selection outputs, and the content selector updates its state based on previously generated tokens.

In the field of dialogue systems, RNN LM generation has been first used for response generation in chat-oriented systems. Here, a LM is trained on large-scale conversation data such as movie subtitles or internet discussions (see Section 2.4), and the generation of the system response is conditioned on the previous user utterance: Vinyals and Le (2015) built a basic chat system using the seq2seq approach. Here, a RNN encoder is applied to the user utterance to obtain the initial state for the RNN LM that generates the response. The seq2seq technique is also used by Li et al. (2016a) and Li et al. (2016c), who devise improved training methods based on mutual information (Manning and Schütze, 2000, p. 66ff.) and reinforcement learning (Sutton and Barto, 1998).

Since the basic seq2seq approach does not include information from a wider context than the most recent user utterance, it often yields an incoherent conversation. Several works have addressed this issue in different ways: Working with Twitter conversations and movie transcripts, Li et al. (2016b) model different speakers in a vector space and insert speaker vectors as additional inputs into the generating LM. Luan et al. (2016) use a single RNN LM for the whole conversation to increase coherence, and they add latent role models based on different vocabulary of users asking and responding in a technical discussion forum. Serban et al. (2016) use a hierarchical setup with two encoder RNNs, one for the current user utterance, and another one to compress the whole dialogue history so far. Xing et al. (2016) enrich the RNN LM setup with explicit topic modeling; they add an encoder RNN over topical words, and a feed-forward NN as a topic summarizer.

The chat-oriented systems have been greatly improved using RNN LMs and can be trained with vast amounts of plain text data without any annotation, but

their outputs cannot be controlled explicitly; therefore, they cannot be applied directly in a task-oriented setting, which is the subject of this thesis. However, some of the methods used in these works are applicable to task-oriented NLG.

In task-oriented SDSs, RNN LM generation has been first used by Wen et al. (2015b). They sample sentences from one RNN LM and use a second RNN working in a reversed direction to rerank the output, along with a convolutional NN reranker. The LM uses a one-hot DA encoding as an additional input into each step. An extension of this work in Wen et al. (2015a) then features an improved RNN architecture which only requires DA input in the first step; it is propagated through the network and adjusted based on what has been generated so far. This version also drops the convolutional reranker. Sharma et al. (2016) present a further improvement to the setup by adding a seq2seq-style encoder. Wen et al. (2016c) experiment with domain adaptation in their Wen et al. (2015a) system by creating fake in-domain data and using discriminative training to fine-tune the generator parameters.

A RNN LM based generator is also a part of the recent end-to-end task-oriented dialogue system of Wen et al. (2016a). Here, the system directly generates a response given a user utterance as in chat-oriented systems, but it tracks the dialogue state explicitly. The encoder in this system integrates the language understanding and dialogue manager modules of a SDS, and the generation RNN has access in each step to a RNN-encoded user utterance, an explicitly modeled dialogue state, and an "output action" vector computed on top of the previous inputs. Wen et al. (2016b) further experiment with different generator RNN architectures and improved training techniques.

## 2.4 NLG Training Datasets

The number of publicly available datasets suitable for NLG experiments is rather small, compared to other areas of NLP, such as MT, where both large quantities of training data and evaluation datasets are published every year in connection with the Workshop on Statistical Machine Translation (WMT) shared tasks (e.g., Bojar et al., 2015, 2016c),[4] or syntactic parsing, where corpora for many different languages have been made available by the Conference on Natural Language Learning (CoNLL) competitions (Buchholz and Marsi, 2006; Nilsson et al., 2007; Hajič et al., 2009) and the HamleDT (Zeman et al., 2012,

[4]See `http://www.statmt.org/wmt16/` (Accessed: March 3, 2017) and analogous websites for the years 2006 through 2017. Since the year 2016, the workshop has been renamed as Conference on Machine Translation.

2014) and Universal Dependencies (Nivre et al., 2016)[5] projects. Moreover, NLG datasets have been typically only released on the authors' webpages. As the authors change their positions or redesign their webpages, some datasets become unavailable over time.

Experimenting on publicly available datasets or publishing new sets for experiments has been more common in text-based NLG than in NLG for SDSs; there are a few text-based NLG datasets available which have been used in multiple experiments. SumTime-Meteo (Sripada et al., 2003; Reiter et al., 2005) is a dataset of raw weather data and their corresponding structured textual descriptions containing 1,045 items.[6] Out of these data, Belz and Kow (2010a) selected 483 wind speed forecasts for their Prodigy-Meteo set.[7] Wong and Mooney (2007) created the GeoQuery and RoboCup datasets,[8] which feature semantic tree representations and corresponding sentences in the domain of geographic trivia questions and sports commentary, respectively. The former contains 880 examples, the latter only 300. Liang et al. (2009) describe the probably largest public NLG dataset called WeatherGov,[9] containing over 29,000 weather forecasts along with the corresponding data events and fine-grained alignments. Konstas and Lapata (2013) used the ATIS flight information corpus for SDSs (Dahl et al., 1994) to regenerate customer requests from semantic parses, and they published the resulting "reversed" dataset with around 5,000 natural language search queries and their meaning representations.[10] Most of the full textual NLG sets assume a content selection step, which is not used in our work.

Several datasets are available especially for the NLG subtask of referring expression generation (van Deemter et al., 2006; Viethen and Dale, 2008; Belz and Kow, 2010b); some of them were used in the Generation Challenges shared tasks.[11]

---

[5] `http://universaldependencies.org/` (Accessed: June 30, 2016). This project also includes converted data from the previous projects.

[6] The dataset used to be available at `http://www.csd.abdn.ac.uk/research/sumtime`, but the link appears to be dead as of June 30, 2016.

[7] `https://sites.google.com/site/genchalrepository/data-to-text/prodigy-meteo` (Accessed: June 30, 2016).

[8] `http://www.cs.utexas.edu/users/ml/wasp/` (Accessed: June 30, 2016).

[9] `https://cs.stanford.edu/~pliang/data/weather-data.zip` (Accessed: June 30, 2016).

[10] `http://homepages.inf.ed.ac.uk/ikonstas/index.php?page=resources` (Accessed: June 30, 2016).

[11] `https://sites.google.com/site/genchalrepository/` (Accessed: June 30, 2016).

Publicly available corpora for NLG in SDSs have been up until now very scarce. The SPaRKy restaurant recommendation corpus (Walker et al., 2007)[12] contains just 20 alternative realizations for each of 15 different detailed text plans; on the other hand, each of them typically spans several sentences. The corpus and the corresponding sentence planner (see Section 2.2) focuses mainly on sentence aggregation according to rhetorical structures and is closely tied to the Meaning-Text formalism (Melčuk, 1988).

Mairesse et al. (2010)[13] published a dataset of restaurant recommendations where each of the 202 distinct input DAs is accompanied by two different textual paraphrases in the form of one or two sentences, i.e., the set contains 404 items in total. It also includes detailed manual alignments between words and phrases in the paraphrases and DA items. The DAs feature 9 different slots (*food*,*area*,*pricerange*, etc.), which may be repeated; "non-enumerable" values such as restaurant names or phone numbers have been delexicalized (replaced by an "X" symbol, see Section 3.3) to curb data sparsity.

Wen et al. (2015b,a) present two similar sets for restaurant and hotel information domains, both containing over 5,000 DA-sentence pairs.[14] The sets are not distributed with delexicalized slot values (see Section 3.3), but delexicalization is relatively simple to perform and Wen et al. use it where possible[15] in their experiments. The number of distinct delexicalized DAs is much smaller than the set size, 248 for the restaurant domain and 164 for the hotel domain. There are 8 different DA types (*inform*, *confirm*, *request*, etc.) and 12 slots for both domains, 9 of which are shared. The datasets do not include detailed alignment of DA items to phrases, and slots in the same DA cannot be repeated. Similar but larger and more diverse datasets for different domains have been released recently by Wen et al. (2016c), who focus on domain adaptation.[16] The sets contain over 13,000 and over 7,000 DA-sentence pairs in the domains of laptop and TV recommendation, respectively. There is much larger variation within the sets as all DAs are distinct (all possible DA type and slot combinations are exhausted). The domains themselves are also larger, with 14 DA types, 19 slots in the laptop domain, and 15 slots in the TV domain. These datasets are probably the largest available so far for NLG in SDSs.

[12] `http://users.soe.ucsc.edu/~maw/final_out.tar.gz` (Accessed: June 30, 2016). See also the description for Howcroft et al. (2013)'s experiments at `http://www.ling.ohio-state.edu/~mwhite/data/enlg13/` (Accessed: June 30, 2016).

[13] `http://farm2.user.srcf.net/research/bagel/` (Accessed: June 30, 2016).

[14] `https://www.repository.cam.ac.uk/handle/1810/251304` (Accessed: July 1, 2016).

[15] This is not possible for slots such as *kids_allowed* that can only take binary *yes*/*no* values, or the value *dont_care* in several other slots. These values do not appear verbatim in the sentence, but influence its structure (e.g., by verbal negation).

[16] `https://www.repository.cam.ac.uk/handle/1810/255930` (Accessed: July 1, g2016).

There are several SDS datasets available with transcripts of human-human or human-computer dialogues (Dahl et al., 1994; Jurčíček et al., 2005; Georgila et al., 2010; Brennan et al., 2013; Williams et al., 2013; Henderson et al., 2014, and others). However, they are usually not well suited for generation as the data are mostly focused towards language understanding and dialogue management: Either the system responses are produced automatically (using handcrafted NLG) by a real SDS or a human imitating an SDS in a Wizard-of-Oz setup, or the corpus lacks the required fine-grained semantic annotation to be used as generation inputs.

More closely related to our work are large-scale datasets of unstructured dialogues for chat-oriented systems (Danescu-Niculescu-Mizil and Lee, 2011; Lowe et al., 2015)[17] as they include natural replies of both parties in the dialogue. They are much larger than any published NLG datasets; however, they contain no semantic annotation, provide no explicit way of controlling the dialogue flow, and still are not directly applicable to task-oriented SDSs.

[17] Cf. the survey of Serban et al. (2015, p. 21) for more details.

# 3

# Decomposing the Problem

This chapter provides a methodological background for all our experiments in Chapters 4 through 8: it is concerned with a closer definition of the task that we are solving, as well as with defining some of the basic aims and features common to all NLG systems developed in the course of this thesis.

As explained in Chapter 1, the task of NLG in a SDS is to convert the output of the dialogue manager, i.e., some kind of a domain-specific shallow MR, to an utterance in a natural language, typically one sentence. In our work, we use a variant of dialogue acts (DAs) as our MR, which we describe in detail in Section 3.1.

Sections 3.2 and Section 3.3 are concerned with the training data format used by our generators. The former explains their ability to use just pairs of DAs and sentences as training data, without additional fine-grained semantic alignments, as required by previous work. The latter section then details delexicalization, a simple data preprocessing technique employed to address data sparsity.

The following two sections of this chapter discuss the option of separating our NLG process into two stages along the traditional pipeline: sentence planning and surface realization. In Section 3.4, we explain our decision to evaluate and compare a joint, one-step NLG setup with a traditional two-step pipeline. We then introduce our choice of intermediate data representation formalism for the latter approach, deep syntax structures in the form of simplified tectogrammatical trees (t-trees), which are further described in Section 3.5.

The final Section 3.6 provides details on NLG evaluation methods, stressing those that are applied in the experimental chapters of this thesis.

inform(name=X, type=placetoeat, eattype=restaurant, area=riverside,
food=Italian)
*inform(name=X)&inform(type=placetoeat)&inform(eattype=restaurant)*
*&inform(area=riverside)&inform(food=Italian)*

confirm(departure_time="6:00pm")&request(from_stop, to_stop)
*confirm(departure_time="6:00pm")&request(from_stop)&request(to_stop)*

Figure 3.1: A comparison of DAs used throughout the literature (top line of each pair) and our functionally equivalent representation (bottom, in italics, null slot values not shown).

## 3.1 The Input Meaning Representation

Throughout our experiments in this thesis, we use a version of the DA meaning representation from the Alex SDS framework (Jurčíček et al., 2014). Here, a DA is simply a list of triplets (DA items or DAIs) in the following form:

- *DA type* – the type of the utterance or a dialogue act per se, e.g., *hello*, *inform*, or *request*.
- *slot* – the slot (domain attribute) that the DA is concerned with. The range of possible values is domain-specific, e.g., *from_stop* or *departure_time* for public transport information and *food* or *price_range* for restaurant information.
- *value* – the particular value of the slot in the DAI; this is also domain-specific. For instance, possible values for slot *food* may be *Chinese*, *Italian*, or *Indian*.

The latter two members of the triplet can be optional (or null). For instance, the DA type *hello* does not use any slots or values, and the DA type *request* uses slots but not values since it is used to request a value from the user.

This representation is functionally equivalent to that of Young (2009),Young et al. (2010), Mairesse et al. (2010), Wen et al. (2015a) and others, where a DA contains a DA type, followed by a list of slots and values. To convert into our representation, one only has to repeat the same DA type with each slot-value pair (SVP). A comparison of our representation and Young et al. (2010)'s version of DAs is shown in Figure 3.1.

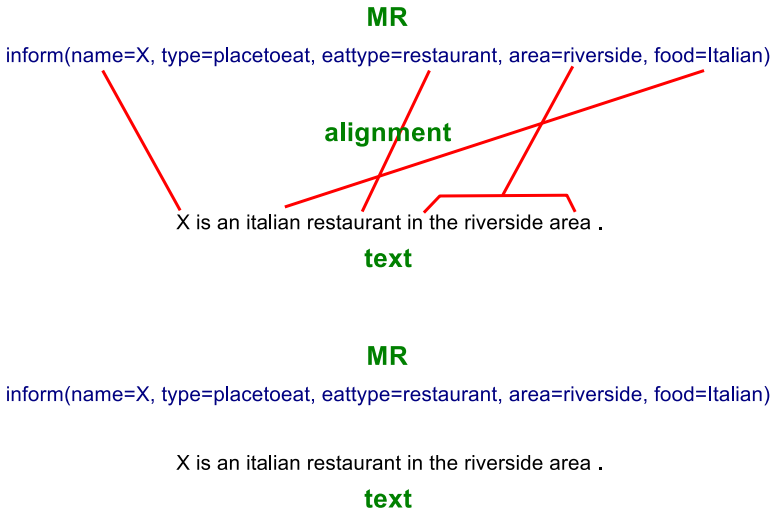


Figure 3.2: Training data for NLG from dialogue acts, with manual fine-grained alignments (top) and without them as used in our experiments (bottom).

For brevity, we continue to show the common DA representation (with one DA type and multiple slots and values) in examples and figures in the further text.

## 3.2 Using Unaligned Data

In all our experiments in this thesis, we use *unaligned* pairs of input DAs and output sentences. This simplifies training data acquisition: Previous NLG systems usually required a separate training data alignment step (Mairesse et al., 2010; Konstas and Lapata, 2013), and this is now no longer required since our sentence planners learn alignments jointly with learning to generate (see Figure 3.2). In addition, alignments learned jointly with generation do not need to be modeled explicitly by hard decisions. They stay latent, which better reflects the fact that correspondences between the elements of the input MR and words or phrases are not always straightforward (see Figure 3.3).

We do not use alignments even in our experiments on the BAGEL dataset (Mairesse et al., 2010), where manual alignments are included in the data (see Chapters 5 and 6).

inform(name=X-name, type=placetoeat, **area=centre**, eattype=restaurant, near=X-near)
*The X restaurant is **conveniently** located near X, **right in the city center**.*

inform(name=X-name, type=placetoeat, **foodtype=Chinese_takeaway**)
*X serves **Chinese food** and has a **takeaway** possibility.*

inform(name=X-name, type=placetoeat, **pricerange=cheap**)
***Prices** at X are **quite cheap**.*

Figure 3.3: Example instances where alignment of dialogue act items to phrases in the sentence is not straightforward.

Words that are aligned to the emphasized DA item do not form a contiguous phrase in any of the examples. The alignment of words "conveniently" from the top example and "quite" from the bottom example to the emphasized DA items is not clear-cut.

## 3.3 Delexicalization

Delexicalization is the replacing of some DA slot values, such as restaurant names or time constants, with placeholders (see Figure 3.4). The generator then only works with these placeholders (both in DAs as well as texts), and they are replaced with concrete values in a simple postprocessing stage. This helps to reduce data sparsity issues and improves generalization to unseen slot values since the possible number of values for some slots is unbounded in theory, and most values are only seen once or never in the training data.

Note that delexicalization is different from using full, fine-grained semantic alignments (see Section 3.2) and can easily be obtained automatically using simple string replacement rules as the values to be delexicalized occur verbatim in training data (possibly in an inflected form for Czech, see Chapter 8).

In our experiments on the BAGEL dataset of Mairesse et al. (2010) described in Chapters 5 and 6, we adopt the partial delexicalization scenario used by the original authors. Here, slot values are divided into "enumerable" (food type and venue type, some neighborhood names) and "non-enumerable" (restaurant, landmark, and certain neighborhood names, postcodes, phone numbers); non-enumerable slot values are then replaced with placeholders and enumerable ones are kept as-is. For our further experiments on the dataset of Wen et al. (2015a) in Section 6.5 and our own collected datasets in Chapters 7 and 8, we use a full delexicalization approach in line with Wen et al. (2015b,a) where all

inform(name="Gourmet Burger Kitchen", type=placetoeat,
eattype=restaurant, area="city centre", near="Tatties (Trinity Street)",
food="Cafe food", food=English)
*Gourmet Burger Kitchen is an English and cafe food restaurant in the city centre near Tatties (Trinity Street).*
inform(name=**X-name**, type=placetoeat, eattype=restaurant,
area="city centre", near=**X-near**, food="Cafe food", food="English")
**X-name** is an English and cafe food restaurant in the city centre near **X-near**.

inform(direction="Fulton Street", from_stop="Rockefeller Center",
line=M11, vehicle=bus, departure_time=11:02am)
*Take line M11 bus at 11:02am from Rockefeller Center direction Fulton Street.*
inform(direction="**X-direction**", from_stop="**X-from_stop**", line="**X-line**",
vehicle="**X-vehicle**", departure_time="**X-departure_time**")
Take line **X-line X-vehicle** at **X-departure_time** from **X-from_stop** direction **X-direction**.

Figure 3.4: Delexicalization examples (top: partial delexicalization in the BAGEL dataset, bottom: full delexicalization in our own public transport information dataset).

Lines in each of the examples from top to bottom: lexicalized DA, lexicalized sentence, delexicalized DA, delexicalized sentence. Placeholders in delexicalized items are highlighted.

slot values appearing verbatim in the text are replaced for generation.[1] For Czech, there is often more than one lexicalization option and concrete values can influence the shape of the whole sentence. Therefore, we also experiment with lexically-informed generation and several lexicalization scenarios (see Section 8.3).

## 3.4 Separating the Stages

We will explore both approaches to NLG sketched in Section 2.1 in this thesis: two-step generation with separate sentence planning and surface realization steps and joint, end-to-end, one-step direct generation. We believe that both

[1] Excluded are values such as *dont_care* or slots with *yes/no* values, which influence the structure of the output sentence, but do not appear in it verbatim.

avenues have their own advantages and disadvantages (see Section 2.1 for details), and that both of them should be explored.

We identified three desired properties of the surface realizer that attempt to mitigate the possible drawbacks of the two-step option: First, it should be conceptually simple so that the two-step setup is not overly complicated. Second, its input sentence plan representation should be easy to generate. And finally and most importantly, using the two-step setup should not require additional training data annotation – the whole generator should still be trainable from pairs of input meaning representation and output sentences. As training sentence plans are required for a separate sentence planner module, these must be obtainable using automatic processing only, i.e., we need an automatic parser from plain text sentences into sentence plans.

We opted for using sentence plans in the form of simplified deep syntactic trees (tectogrammatical trees or t-trees) based on the Functional Generative Description (FGD; Sgall et al., 1986). The t-trees representation (see Section 3.5) is used in the Prague Dependency Treebanks family (PDTs; Hajič et al., 2006, 2012; Bejček et al., 2012), in the CzEng Czech-English parallel corpus (Bojar et al., 2012, 2016a), and in the TectoMT translation system (Žabokrtský et al., 2008; Dušek et al., 2012). It conforms very well to the desired realizer properties stated above:

First, there is a Czech surface realizer available for English-Czech translation in TectoMT which shows that realization from t-trees is viable and very straightforward: It is a simple pipeline of procedural modules operating over a tree API, adding nodes and modifying their properties to transform the deep syntax representation into a surface sentence. It is deterministic, robust, domain-independent, and well-tuned as it produces output for any input and only accounts for 3% of errors in the whole TectoMT system pipeline (Popel and Žabokrtský, 2009). Second, the t-trees representation is a projective[2] labeled dependency tree, a representation for which efficient algorithms exist (Kübler et al., 2009, p. 16ff.). And third, domain-independent automatic parsers into t-trees for several languages are included in the Treex NLP framework (Popel and Žabokrtský, 2010);[3] the English pipeline used to prepare training data for our sentence planners involves a statistical part-of-speech tagger (Spoustová et al., 2007) and a dependency parser (McDonald et al., 2005), followed by a

[2]In projective dependency trees, each word with its descendants forms a contiguous substring of the sentence (McDonald et al., 2005). While in theory, t-trees are always projective, this condition might not be fulfilled in t-trees resulting from automatic analysis (with surface word order). As non-projective t-trees only occur rarely, we assume projective t-trees throughout this thesis.

[3]See `http://ufal.mff.cuni.cz/treex` (Accessed: July 31, 2016).

rule-based conversion to deep syntax trees.[4] The Czech pipeline is similar, comprising of a part-of-speech tagger by Straková et al. (2014), a dependency parser by Novák and Žabokrtský (2007), and subsequent rule-based conversion.[5]

There are two additional advantages of selecting the t-tree formalism as a sentence plan format and using a TectoMT-style pipeline realizer: The first one is reusability: We can simply reuse the Czech t-tree realizer from TectoMT in our experiments with Czech (see Chapter 8), and vice versa – we create a similar realizer for English, which can be reused in TectoMT for translating into English (see Chapter 4). Second, a (mostly) rule-based syntactic realizer allows us to ensure grammatical correctness of the output sentences at all times; in a statistical approach, this may be more difficult.[6]

There are of course several other possible alternatives to the FGD-based t-tree sentence plan representation, such as AMR (Banarescu et al., 2013), CCG (or other formal grammars Steedman, 2000), or Meaning-Text theory based deep syntax (Melčuk, 1988). We chose t-trees over them out of several reasons: The former two representations are more abstract than t-trees, which would result in a more complicated and less controllable surface realizers; in addition, tree structures promise a simpler processing than general graphs or predicate-argument structures with variables. The Meaning-Text deep syntactic representation is quite similar to t-trees, and we did not see significant advantages in using it instead of t-trees; t-trees were chosen over Meaning-Text structures based on the ready availability of processing tools at our department and our familiarity with the formalism.

## 3.5 *t-trees*: Deep Syntax Representation

The t-tree sentence plan structure is represented by deep-syntactic dependency trees that only contain nodes for content words (nouns, full verbs, adjectives, adverbs) and coordinating conjunctions (see Figure 3.5). The nodes maintain surface word order. Each node has several attributes:

- *t-lemma* (or *deep lemma*) – this is typically the base morphological form of the content word for the node, i.e., identical with the surface lemma of

[4] The main Treex module for the English pipeline can be found at `https://github.com/ufal/treex/blob/master/lib/Treex/Scen/Analysis/EN.pm` (Accessed: November 2, 2016).

[5] The main module for the Czech pipeline can be found at `https://github.com/ufal/treex/blob/master/lib/Treex/Scen/Analysis/CS.pm` (Accessed: November 2, 2016).

[6] As shown by our experiments in Chapters 6 and 8, this turned out to be a non-issue for our domains and languages.

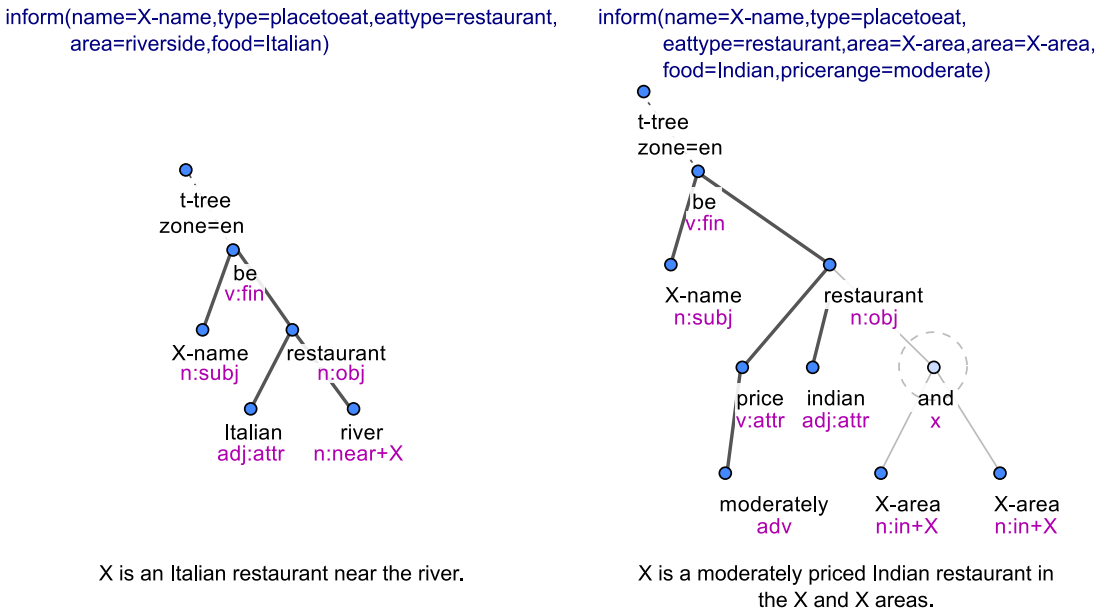


Figure 3.5: Example t-trees (in the middle), with the corresponding DAs (top) and natural language paraphrases (bottom).

Only the two most important node attributes for our experiments are shown: t-lemma (1st line, in black) and formeme (2nd line, in purple).

the content word (e.g., "river" and "be" are the t-lemmas for the nodes representing the phrases *near the river* and *is* in Figure 3.5 on the left).

There are several exceptions to this rule, most notably the following ones:

- Particles of English phrasal verbs are included in the t-lemma (e.g., "break_down").
- Personal pronouns and negation particles use special t-lemmas starting with "#" ("#PersPron" and "#Neg", respectively).

- *functor* – a deep-syntactic/semantic role label. The FGD theory distinguishes five main verb-specific obligatory arguments – actor (ACT), patient (PAT), addressee (ADDR), origin (ORIG), and effect (EFF). In addition, there are about 50 optional adjuncts distinguished based on semantic criteria, such as time (TWHEN), location (LOC), or cause (CAUS).[7]

- *grammatemes* – a list of attributes describing grammatical meaning that cover phenomena such as number, tense, definiteness, or modality. The

[7] A more detailed description of functors with a full list of values can be found at `https://ufal.mff.cuni.cz/pcedt2.0/en/functors.html` (Accessed: August 11, 2016).

set of grammatemes mostly copies the corresponding grammar categories (e.g., present, past, future tense; modality categories correspond to modal verbs).[8]

- *formeme* – a concise description of the surface morphosyntactic form of the node, consisting of its syntactic part-of-speech (where nominal usages of adjectives count as nouns, etc.), prepositions and/or subordinate conjunctions (if applicable), and the form or syntactic position of the node.

  For example, n:subj stands for a noun in the subject position, n:in+X for a prepositional phrase with the preposition *in*, v:fin for a verb heading a finite clause, and v:of+ger for a gerund in a prepositional phrase.[9]

  The formeme attribute was neither a part of the FGD theory nor of the gold-standard PDT annotation. It was added into the automatic t-tree analysis pipeline as a more robust replacement of functors in the TectoMT deep transfer (Žabokrtský et al., 2008; Dušek et al., 2012): Unlike functors, formemes can be assigned on top of dependency parsing via a set of simple rules, and they preserve the morphosyntactic information required for generation.

The t-lemma and formeme attributes are the most important ones for our experiments. Same as in the TectoMT system, we only use functors to mark specific nodes such as coordination, and we do not use functors at all in sentence planning. Grammatemes are required in the surface realizer (see Section 4.1), but we do not use them in sentence planning to simplify our experiments; we only store the most frequent grammateme values for each particular node type (t-lemma and formeme) in the training data and use these values as input to the sentence planner. This is sufficient for our experiments on the BAGEL restaurant information data of Mairesse et al. (2010) in Chapters 5 and 6. For the larger and more variable domain of our Czech restaurant dataset in Chapter 8, we needed to explicitly include two of the grammatemes in the representation – person for personal pronouns and sentence modality (sentence type).[10]

[8] A detailed description of grammatemes can be found at `https://ufal.mff.cuni.cz/pcedt2.0/en/grammatemes.html` (Accessed: August 11, 2016).

[9] For a detailed description of formemes, see (Dušek et al., 2012).

[10] In the implementation, they are simply included as parts of the t-lemma and formeme, respectively (cf. Figure 8.3 and Footnote 11 in Chapter 8).

## 3.6 Evaluation Metrics

Unlike in NLP tasks involving, e.g., morphological or syntactic annotation, where a direct comparison to a single ground truth value is typically used to assess the performance of an automatic system (Resnik and Lin, 2010), there are usually many possible valid outputs for NLG. Several different approaches have been applied to evaluating NLG (Hastie and Belz, 2014; Gkatzia and Mahamood, 2015): intrinsic, further divided into automatic scores and human ratings, and extrinsic, such as users' success in completing a task based on information provided by the NLG output. In this work, we limit ourselves to evaluating our systems intrinsically; extrinsic evaluation in a live dialogue system is left for future work.

The problem of intrinsic NLG evaluation is very similar to the situation in MT, except for the fact the the "source language" here is a formal MR: There is a natural language sentence produced by a system and one or more human-authored references for a test set. Therefore, MT evaluation methods are generally applicable to NLG.

As mentioned above, intrinsic evaluation metrics NLG (and MT) can be divided into automatic comparisons against human-created references and manual human assessments. Automatic NLG metrics are typically based on word-by-word comparisons against reference texts, measuring word overlap. This approach is cheap and scores can be obtained instantly, thus allowing for usage in system tuning. However, their capability of assessing all possible output variants adequately is limited and their correspondence to human judgments has been disputed (Stent et al., 2005; Callison-Burch et al., 2006). Correlation of the scores produced by the metrics with human assessment is typically low on segment (sentence) level (Lavie and Agarwal, 2007; Chen and Cherry, 2014). However, overall corpus-level correlation tends to be fairly high (Papineni et al., 2002; Coughlin, 2003; Galley et al., 2015; Bojar et al., 2016b).

Human evaluation provides a much more accurate estimate of an NLG system's performance; however, it is often a rather lengthy and expensive process. Both approaches are therefore combined in practice: automatic metrics are used in system development and tuning, but they are supported by human evaluation for the final version of a system.

In the following, we mention several automatic and human NLG metrics in particular, focusing mainly on the metrics used in our experiments in Chapters 5, 6, 7, and 8. For automatic word-overlap based metrics, we describe in more detail BLEU and NIST, two of the oldest and arguably the most frequently used metrics in NLG. There have been many more word-overlap metrics de-

veloped for MT (cf. e.g., Stanojević et al., 2015; Bojar et al., 2016b), but their superiority to BLEU has been disputed (Graham and Baldwin, 2014) and their application in NLG has so far been limited. We also list an additional, different automatic metric used exclusively for delexicalized NLG from DAs (including our work), the *slot error rate* (ERR) based on counting DA slot value placeholders in the output. In human evaluation metrics, we briefly describe and categorize the most widely used methods and finish with a note on the methods used in this thesis.

## Automatic NLG Evaluation

**BLEU.** The most frequently used automatic metric in NLG is the same as the de-facto standard for MT evaluation, BLEU score (Papineni et al., 2002). It is based on comparing $n$-grams of tokens from the generated output against human-created reference paraphrase(s) for the input MR. The score is computed as a geometric mean of $n$-gram precisions lowered by a brevity penalty factor, according to the following formula:

$$\text{BLEU} = \exp\left(\sum_{n=1}^{N} \frac{1}{N} \cdot \log(p_n) - \max\left\{\frac{L^*_{\text{ref}}}{L_{\text{sys}}} - 1, 0\right\}\right) \tag{3.1}$$

In (3.1), $L_{\text{sys}}$ denotes the number of tokens in the system output, and $L^*_{\text{ref}}$ stands for the number of tokens in the reference text that is closest in length to the system output. The $n$-gram precision $p_n$ is computed as:

$$p_n = \frac{\sum_{\text{segment}} \text{\# of matching } n\text{-grams in segment}}{\sum_{\text{segment}} \text{\# of } n\text{-grams in segment}} \tag{3.2}$$

The standard value of $N$ (highest $n$-gram length considered) for BLEU is 4; however, other variants of BLEU-$N$ are also used (typically for $N < 4$).

BLEU is very simple to compute; a fact which contributed to its wide adoption. On the other hand, it is unable to give credit for synonyms (and different inflection forms of the same word) not covered by reference outputs or weigh the importance of individual $n$-grams (where content words such as nouns are arguably more important than prepositions or punctuation tokens). This led to questions about BLEU's adequacy and inspired development of other, more sophisticated metrics (e.g., Lavie and Agarwal, 2007). However, none of the other metrics has gained as wide adoption as BLEU; they are mostly used to supplement it.

| | |
|---|---|
| Input DA | iconfirm(alternative="next")&inform(duration="**X-duration**", departure_time="**X-departure_time**") |
| Reference | The following connection is at **X-departure_time** and will take **X-duration** minutes. |
| Output A | The next **X-vehicle** is at **X-departure_time**. [**X-duration**] |
| Output B | The ride is at **X-departure_time** and takes **X-duration** minutes. [next] |

Table 3.1: Slot error rate (ERR) example.

Both the input DA and the outputs are partially delexicalized (values for the slots duration and departure_time are replaced with placeholders, but the slot alternative uses specific values). Slot value placeholders are shown in bold, semantic errors are marked in color: superfluous, missing.
ERR for Output A is $(1+1)/2 = 1$ (one missing and one additional placeholder, two placeholders in the input DA).
ERR for Output B is $(0+0)/2 = 0$ (no missing or additional placeholders). Note that Output B misses an indication of the next connection. Since the slot alternative is not delexicalized, this is not captured by the ERR metric.

**NIST.** The dominance of BLEU is even greater in NLG than in MT; with the only other metric used more widely in NLG being NIST (Doddington, 2002). NIST attempts to alleviate some of the possible problems of BLEU. It uses arithmetic average instead of geometric average for the $n$-gram precisions, lowering variance for low precision in higher $n$-grams. It also applies an adjusted brevity penalty formula that is more strict for very short translations than BLEU's brevity penalty. And most importantly, $n$-grams are weighted according to their information value. The NIST score is computed as follows:

$$\text{NIST} = \sum_{n=1}^{N} \frac{\sum_{\text{matching } n-\text{grams}} \text{info}(n\text{-gram})}{\sum_{\text{all } n-\text{grams}} 1} \cdot \exp\left(\beta \log^2 \min\{\frac{L_{\text{sys}}}{\overline{L}_{\text{ref}}}, 1\}\right) \quad (3.3)$$

The information value of an $n$-gram $w_1, \ldots, w_n$ in (3.3) is computed over the reference corpus as:

$$\text{info}(w_1, \ldots, w_n) = \log_2 \frac{\text{\# of occurrences of } w_1, \ldots, w_{n-1}}{\text{\# of occurrences of } w_1, \ldots, w_n} \quad (3.4)$$

The variables $L_{\text{sys}}$ and $\overline{L}_{\text{ref}}$ in (3.3) denote the number of tokens in the system output and the average number of tokens in the reference texts, respectively. The value $\beta$ is chosen so that the brevity penalty equals 0.5 if the system output is 1/3 shorter than the average reference. The standard value of $N$ is 5 for NIST.

**ERR.** A different kind of automatic metric is the *slot error rate* (ERR) used by Wen et al. (2015a, 2016c): here, the generated outputs are not compared against the reference but against the input DA. The ERR metric merely checks for the presence for DA slot placeholders (delexicalized slot values, see Sections 2.4 and 3.3) in the output. This is only applicable to DA slots that have been delexicalized (i.e., only the slots whose values will appear verbatim in the output); however, the ERR metric still gives at least a rough estimate of how well the generated outputs reflect the input meaning representation. The ERR metric is computed as follows:

$$\text{ERR} = \frac{M + A}{S} \tag{3.5}$$

In (3.5), $S$ denotes the total number of delexicalized slots in the input DA, and $M$ and $A$ denote the number of missing and superfluous slot placeholders in the generator output, respectively (see Table 3.1 for an example).

## Human Metrics

For intrinsic human NLG evaluation, there is no single standard and each study typically designs its own method. A majority of the recent works employ "user like" evaluation, i.e., subjective human assessments on a (usually 5- to 7-point) Likert scale (Hastie and Belz, 2014; Gkatzia and Mahamood, 2015). In a typical case, two different criteria are used, roughly corresponding to *adequacy* (accuracy in reflecting the input MR in the output text) and *fluency* (quality of phrasing/expression in the outputs). The exact definitions and labels vary, e.g., Reiter and Belz (2009) use "accuracy" and "clarity" and Mairesse et al. (2010) denote the two measures as "informativeness" and "naturalness", but the basic distinction is the same. Some works use additional, task-specific criteria, such as correspondence to personality traits (Mairesse and Walker, 2011) or to the overall goal of the text (Kiddon et al., 2016).

While some authors show their system outputs to human raters in isolation (Stent et al., 2005; Mairesse and Walker, 2011; Konstas and Lapata, 2013), many prefer to display together several outputs of different system variants for the same input MR (Bangalore et al., 2000; Reiter and Belz, 2009; Mairesse et al., 2010; Lampouras and Vlachos, 2016), thus allowing direct comparisons among the systems and shifting the rating from absolute towards relative scales.

The distinction between fluency and accuracy is mainstream in NLG evaluation, but it has been disputed in MT literature. In comparison to a single, overall quality judgment, split criteria tend to produce less consistent ratings

(Callison-Burch et al., 2007; Koehn, 2010, p. 220). Some studies have also shown that the two criteria are hard to separate: ratings along both scales tend to be closely related (Nenkova et al., 2010; Koehn, 2010, p. 220). The field of MT therefore largely moved towards evaluation according to overall quality only. Furthermore, relative comparisons/rankings are used rather than absolute Likert scale judgments (e.g., Bojar et al., 2015, 2016c). Several recent NLG studies also prefer to use an overall quality criterion, either to complement fluency/adequacy ratings (Wen et al., 2015b,a; Manishina et al., 2016), or as the only human measure (Sharma et al., 2016). Moreover, Wen et al. (2015b,a) use explicit pairwise preference (i.e., relative) rankings in addition to absolute Likert scale assessments.

In our human evaluation studies in Chapters 7 and 8, our main goal is to decide which system variant will provide outputs that are generally most preferred by the users. Therefore, we focus on relative comparisons using a single quality/preference criterion as this promises to be the most efficient way of achieving consistent and unambiguous comparisons. In order to obtain a deeper insight into the different types of errors induced by our generators, we also perform small-scale manual expert judgments of our own in Chapters 5, 6, and 8. Here, we assess the number of objective semantic and grammatical errors of various types, similarly to less used but very elaborate MT evaluation methods described by Popel and Žabokrtský (2009) or Lommel et al. (2013), though in a much more limited extent.[11]

[11] These metrics, in general, categorize and annotate various error types. They provide a very accurate and insightful picture of a system's performance, but require expert judges and are very time-consuming. This is also the reason for our limiting the study to few error types and small data samples.

# 4

# Experiments in Surface Realization

This chapter is an account of our own experiments with surface realization – generating natural language sentences from t-trees (see Section 3.5), with a few exceptions that are mentioned in the text. Our main goal was to create a simple and domain-independent realization module, easy to adapt for further language in the future.

This goal has largely been fulfilled: Based on a similar module for Czech, we developed a new general-domain, mostly rule-based surface realizer for English, which is then used in our experiments with full generation from DAs in Chapters 5 and 6. We describe the structure of this realizer and perform basic evaluation in Section 4.1. Apart from our experiments targeted at NLG in SDS, we also used our realizer as a part of an MT system, which we briefly describe in Section 4.2. We show that the realizer leaves some room for improvement, but in general serves its purposes well. Since its implementation, some of its components have been reused by others to build similar realizers for Spanish and Basque (Aranberri et al., 2016).

The following two sections are dedicated to smaller experiments with the introduction of statistical modules into the realization pipeline. In Section 4.3, we report on our small, proof-of-concept experiment in generating complex verbal groups from t-tree nodes. We achieved a very high performance, but the resulting module was not used in the realizer pipeline since it did not provide enough of an advantage over a rule-based implementation. Section 4.4 describes our extensive experiments in statistical morphological inflection, where our *Flect* system based on logistic regression achieved very good results for six different languages. We integrated Flect into the English surface realizer pipeline, and we show that it improves on a dictionary-based module.

In the final Section 4.5, we offer a summary of the results we obtained, as well as a few concluding remarks.

Parts of this chapter are based on works we published previously: The description of the English realizer and its application to Czech-English machine translation in Sections 4.1 and 4.2 is adapted from (Dušek et al., 2015); Section 4.4 on morphological inflection generation is originally based on (Dušek and Jurčíček, 2013).

## 4.1 Constructing a Rule-based Surface Realizer for English

We have developed a new implementation of an English surface realizer from t-trees within the Treex NLP framework (Popel and Žabokrtský, 2010),[1] which mostly adapts the Treex Czech realizer pipeline modules (Žabokrtský et al., 2008; Popel, 2009, p. 84ff.) and shares their language-independent code components. Our main goals were to reuse as much code and ideas from the Czech realizer as possible and to arrive as fast as possible at an implementation that would work well with our sentence planning experiments (see Chapters 5 and 6).

Although an English surface realizer had been implemented by Ptáček (2008) for the same NLP framework, its code is incomplete and now long obsolete. Therefore, we built all the modules listed below anew, adapting Czech realizer code and separating language-independent parts where applicable.

### Pipeline Description

The resulting new English surface realizer includes all the components necessary to handle surface language phenomena: auxiliary words, inflection, word order, agreement, punctuation, and capitalization. All modules are rule-based, except for morphology generation, which employs a statistical module described in Section 4.4.

At the start of the realization pipeline, a new surface dependency tree[2] is created as a copy of the source t-tree, with surface lemmas copied from deep lemmas (t-lemmas) and dependency labels, word forms, and morphology left undecided. All further changes are performed on the surface dependency

[1] `http://ufal.mff.cuni.cz/treex` (Accessed: August 12, 2016).

[2] Surface dependency trees are called *analytical* or *a-trees* in the FGD terminology.

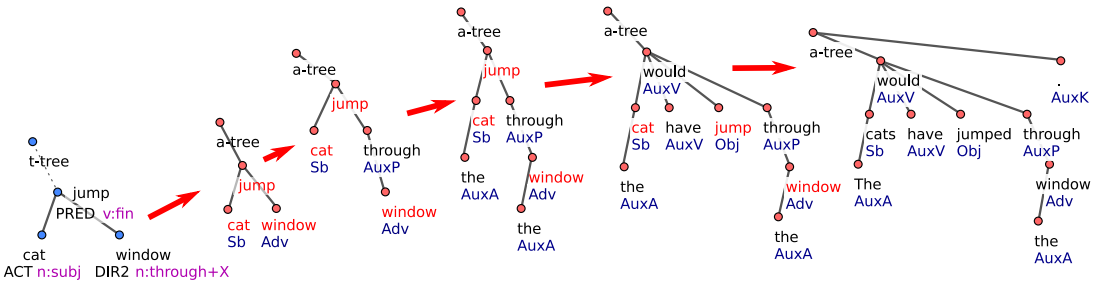


Figure 4.1: Rule-based surface realization pipeline example.

The t-tree for the sentence "The cat would have jumped through the window." is gradually transformed into a surface dependency tree (a-tree). Uninflected words are shown in red in a-trees, dependency labels are shown in blue, pre-filled morphological attributes are not shown.
Partially completed surface dependency trees from the left: (1) morphological attributes are filled in, subjects are marked, word order and agreement are enforced (Steps 1–4 of the pipeline description). (2 and 3) grammatical words are added – preposition and articles (Step 5). (4) auxiliary verbs are added (Step 6). (5) punctuation is added, words are inflected, and sentence start is capitalized (Steps 9–13).

tree, consulting information from the t-layer tree. The pipeline consists of the following steps (see also Figure 4.1 for illustration):[3]

1. Morphological attributes are filled in based on grammatemes.

2. Subjects are marked (to support subject-predicate agreement).

3. Basic English word order for declarative sentences is enforced. This only contains very general rules, such as maintaining the SVO-order or the adjective-noun order, since a pre-ordered input t-tree is expected. Usually no changes in word order are required for postprocessing sentence plans produced by our sentence planners (see Chapters 5 and 6) since these are already ordered. For input t-trees resulting from translation in TectoMT, preliminary tests with source-language ordering from several different languages indicated that this amount is sufficient in most cases, and language-specific word-order transformations are expected to be part of TectoMT's transfer stage.

[3]Since the author of this thesis created the initial version of the realizer, there have been a few minor bugfixes or improvements done by Rudolf Rosa, Zdeněk Žabokrtský, and Michal Novák. The improvements mainly concern word order and auxiliaries in imperative or interrogative sentences (Step 7). Full history of contributions to the realizer can be found at Treex GitHub (`https://github.com/ufal/treex/commits/master/lib/Treex/Block/T2A/EN` and `https://github.com/ufal/treex/commits/master/lib/Treex/Block/A2W/EN`, accessed: August 13, 2016).

4. Subject-predicate agreement in number and person is enforced – predicates have their number and person filled based on their subject(s).

5. Grammatical words are added. These are based on the contents of formemes (prepositions, subordinating conjunction, infinitive particles, possessive markers), grammatemes (negation particles and articles), and deep lemmas (phrasal verb particles).

6. Auxiliaries for compound verb forms are added, expressing the voice, tense, and modality. Auxiliaries are also added for questions and sentences with existential *there*.

7. Imperative subjects are removed, question subjects are moved after the auxiliary verb.

8. Negation particles are added for verbs as well as selected adjectives and adverbs.

9. Final punctuation is added to the end of the sentence and commas are added into coordinations and appositions, after clause-initial phrases preceding the subject, and in selected phrases (based on formemes).

10. Words are inflected based on their lemma and morphological attributes. We use rules for personal pronouns, MorphoDiTa English dictionary (Straková et al., 2014) for unambiguous words, and Flect (Dušek and Jurčíček, 2013, described in Section 4.4 in detail) for all remaining words requiring inflection.[4]

11. The English indefinite article *a* is changed into *an* based on the following word.

12. Repeated coordinated prepositions and conjunctions (added based on formemes in Step 5) are deleted, e.g., *near X and near Y* becomes *near X and Y*.

13. The first word in the sentence is capitalized.

The output sentence is then obtained by just combining all the nodes in the resulting surface dependency tree.

[4] Alternatively, a language model *could* be used to select the word forms (see Section 1.4; cf. also Section 8.3). Flect uses just a short context of neighboring lemmas, but it generalizes also to unseen words (thanks to morphological features).

### Realizer Evaluation

To evaluate the realizer on a very broad domain, we have run a round-trip test similar to the one conducted for Czech in Dušek et al. (2012): We first automatically analyze English texts into t-trees, then run our surface realizer to regenerate texts and evaluate the results using BLEU score (Papineni et al., 2002) against the originals. On texts from the Prague Czech-English Dependency Treebank (PCEDT) 2.0 (Hajič et al., 2012) sections 22 and 23,[5] the English realizer reaches a BLEU score of 77.47%. This score is relatively high given that the original is used as the only reference and even minor deviations are penalized.[6] In fact, most regenerated sentences are nearly identical to the source, with small differences which do not change the meaning, such as *is not* vs. *isn't*. Still, the output leaves ample room for improvement as some sentences are distorted by the transformation.

A closer analysis of a sample of the sentences processed by the round-trip pipeline showed that most serious errors occur in the t-layer analysis part of the pipeline. Incorrect part-of-speech tags or syntactic dependencies then lead to problems on the output, such as incorrect word order or verbal inflection. Most errors due to the realizer occur in punctuation. However, this is not our top priority as English punctuation rules allow a lot variants, and punctuation errors are not perceived as very severe by most readers. Moreover, punctuation has only a minor influence on the text-to-speech conversion in spoken dialogue. Other generation errors occur in negative words, where the dictionary used to create surface lemmas is lacking (e.g., *not fit* is produced as a fallback instead of *unfit* since this form is not included in the dictionary). This problem can be solved to a large part by collecting a larger dictionary, or possibly training the morphological generation (see Section 4.4) to produce negation as well. For our dialogue domains, the dictionary offers reasonable coverage.

## 4.2 Using the Realizer in the TectoMT Translation System

As described in Section 3.4, one of the advantages of t-tree realization is its potential for reuse. Since its implementation, our English realizer has been successfully applied in TectoMT systems translating into English within the

[5] The sentences are originally taken from the Wall Street Journal section of the Penn Treebank (Marcus et al., 1993).

[6] We obtained an even higher score in a similar test on a limited domain used in experiments in Chapter 5 (see Section 5.1).

QTLeap[7] project (Rosa et al., 2015; Popel et al., 2015a). The author of this thesis is responsible for its application in the Czech-English and Dutch-English translation directions. Let us therefore first describe the usage of the realizer in the TectoMT system and then summarize the results of TectoMT evaluation on the two translation pairs mentioned above.

## The TectoMT Pipeline Architecture

The TectoMT system starts with the automatic parsing of source language sentences into t-trees (see Section 3.4) using a hybrid pipeline consisting of statistical tools and smaller rule-based modules. The main parts of the Czech analysis pipeline are the MorphoDiTa part-of-speech and morphological tagger (Straková et al., 2014) and the Maximum Spanning Tree (MST) dependency parser adapted for Czech (Novák and Žabokrtský, 2007). The ensuing conversion of dependency trees into t-trees is performed by a sequence of rule-based modules, with the exception of a linear classifier for functor assignment (Bojar et al., 2016a). For Dutch, part-of-speech tagging, morphology, and dependency parsing are handled by the Alpino parser (Van Noord, 2006). Alpino dependency trees are then converted into t-trees using a rule-based pipeline.

The TectoMT pipeline continues with the cross-lingual transfer at the deep syntactic level. The transfer exploits the fact that t-tree representations of the same sentence in different languages are much closer to each other than surface strings – there is often a 1:1 node correspondence. TectoMT translates t-trees node-by-node, assuming that the tree shape will not change (apart from special cases handled by rules, such as the translation of English noun groups into Dutch compound nouns).

The translation is further split up into individual attributes – t-lemmas (deep lemmas), formemes, and grammatemes are translated separately. Lemma and formeme translation uses an interpolation of discriminative maximum entropy models (Mareček et al., 2010) and simple conditional probability models. Grammateme transfer is rule based; in most cases, grammatemes remain unchanged. For translation into English from Czech and Dutch, the gender grammateme for all semantic nouns except personal pronouns is reset. For Czech-to-English translation, the definiteness grammateme must be set since this grammateme is not present in Czech t-trees (Czech does not have articles or any other grammatical expression of definiteness). We use a rule-based module based on an older English article detection module by Ptáček (2008).

[7] European Commission 7th Framework Program project no. 610516 "Quality Translation by Deep Language Engineering Approaches", see `http://qtleap.eu` (Accessed: October 12, 2016).

| Task | Dutch-English | | Czech-English | |
|---|---|---|---|---|
| | IT | news | IT | news |
| Phrase-based | 25.57 | 23.50 | 19.03 | 24.03 |
| TectoMT | 27.09 | 19.40 | 20.53 | 13.04 |

Table 4.1: BLEU scores for TectoMT translation within the QTLeap project (Pilot 2 system version; Popel et al., 2015a)

The translated t-tree resulting from the analysis and transfer pipeline described above is then fed into our English t-tree realizer.

## Evaluation within the QTLeap Project

TectoMT has been evaluated extensively on translations in the information technology (IT) domain within the QTLeap project, both using automatic metrics and comparative human evaluation with a phrase-based MT baseline (Popel et al., 2015a; Del Gaudio et al., 2015).[8] The Czech-English TectoMT system also competed in the 2015 WMT news translation task (Bojar et al., 2015; Dušek et al., 2015). We include the results from QTLeap IT translation for Dutch-English and Czech-English in Table 4.1, which also shows a comparison with the phrase-based MT baseline.

We can see from Table 4.1 that while TectoMT is able to improve over the phrase-based baseline for both languages in the narrow IT domain, it is lacking in the broader news domain. Superior performance on the IT domain has been confirmed in extrinsic evaluation based on cross-lingual information retrieval (Del Gaudio et al., 2015). On the other hand, the problems on the broader domain showed also in the WMT 2015 news translation task results for Czech-English, where TectoMT trails the table in both automatic scores and human rankings (Bojar et al., 2015).[9]

If we look at the translated sentences in detail, it becomes apparent that most errors are introduced in the analysis and transfer parts of the pipeline, similarly to the round-trip evaluation experiment described in Section 4.1. Dušek et al. (2015) report untranslated words and translation model errors, problems in word reordering for the Czech-English direction, as well as poor performance

[8] Moses (Koehn et al., 2007) is used as the baseline phrase-based system.

[9] The automatic scores table is available online at `http://matrix.statmt.org/matrix/systems_list/1782` (Accessed: October 7, 2016). Dutch-English was not among the language pairs evaluated in the competition.

(1) *Output:* One Council, how into that moment to do: carefully this page snatch and make from it bookmark.

*Source:* Jedna rada, jak se v tu chvíli zachovat: Opatrně tuhle stránku vytrhněte a udělejte si z ní záložku.

*Reference:* *A piece of advice on how to proceed at that moment: gently excise this page and make it your bookmark.*

(2) *Output:* Mr. Englund a historian is swedish and a journalist.

*Source:* Pan Englund je švédský historik a novinář.

*Reference:* *Mr. Englund is a Swedish historian and journalist.*

(3) *Output:* Their lives flikkeren as votiefkaarsen in a church; new is added to the altar other is been.

*Source:* Hun levens flikkeren als votiefkaarsen in een kerk; nieuwe worden toegevoegd aan het altaar terwijl andere worden uitgemaakt.

*Reference:* *Their lives flicker like votive candles in a church; new ones are added to the altar while others are put out.*

(4) *Output:* From the almost beginning, this is an inspiring book.

*Source:* Vrijwel vanaf het begin is dit een bezielend boek.

*Reference:* *Almost from the start, this is a moving book.*

Figure 4.2: Example TectoMT translations from Czech and Dutch into English, with errors highlighted.

Each example shows the TectoMT output, the original Czech or Dutch sentence, and a human translation of the original (based on reference translations of the dataset, corrected). Errors are color-coded:

teal: source parsing errors (cause wrong word order in (2))

purple: t-lemma translation errors (e.g., "rada" is translated as "council", not "advice" in (1))

red: words left untranslated in the output (a specific t-lemma translation error)

magenta: formeme translation errors (e.g., "v" is translated as "into", not "at" which would fit the context in (1))

pink: article assignment errors (the word "bookmark" is missing an article in (1))

olive: word ordering errors caused by transfer (in (1), Czech word order is preserved, which is incorrect for English)

cyan: word ordering errors caused in realizer (in (4), the preposition and the article added by the realizer should follow the adverb "almost")

orange: inflection errors (in (3), they are probably caused by an error in a realizer rule carrying grammar information from t-trees to surface trees)

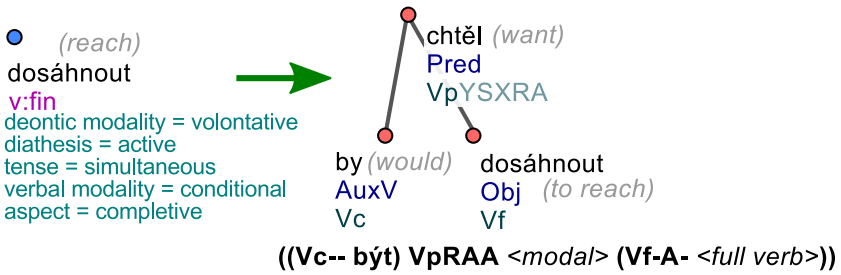


Figure 4.3: Example of the compound verb form prediction.

The source t-tree node (with t-lemma, formeme, and some of the grammatemes) is shown on the left, the target surface dependency structure (word forms, dependency labels and part-of-speech-tags) is on the right, along with the bracketed encoding used by the classifier.

of the article assignment transfer module. We examined manually a sample of Dutch-to-English and Czech-to-English translations and identified the same error sources for Czech as well as problematic parsing of some sentences; for Dutch, translation model errors (especially problems in translating compounds) and parsing errors are the most often encountered issues. The realizer still has some issues with word order and inflection, but they are rarer (cf. Figure 4.2 for examples). This confirms Popel and Žabokrtský (2009)'s results for Czech and shows that the realizer offers a reasonable performance for a wide variety of applications.

## 4.3 Statistical Compound Verb Form Generation

As a proof-of-concept experiment in replacing parts of the rule-based surface realization pipeline by modules trained from data, we have created a simple statistical module that is able to predict compound verbal forms along with their dependency structure. Given a t-lemma (deep lemma) and grammatemes (deep grammatical properties) from the t-tree, auxiliary verbs along with their morphology and dependency structure are predicted. An example of this task is shown in Figure 4.3. This module has not been used in any realizer pipeline since the corresponding rule-based modules work well enough for both Czech and English, but its evaluation on Czech data from the PDT 2.0 (Hajič et al., 2006) shows that replacing parts of the rule-based pipeline by very simple statistical modules is possible (cf. Ballesteros et al., 2014).

We have reformulated the task of verb form generation as a simple multiclass classification problem using a bracketed encoding of the target structures. As shown in Figure 4.3, we replaced t-lemmas of full verbs and modal verbs with special symbols to reduce data sparsity and only use certain properties from the morphological tag of the target inflected verb form in the encoding (verbal form, such as infinitive or participle, as well as tense, voice, and negation).[10] Furthermore, we normalize word order to the most common variant. This gives only 90 different target classes on the Czech PDT 2.0 data.

We use the LIBLINEAR logistic regression classifier (Fan et al., 2008) with the following features: t-lemma, grammateme values, person of the subject in the clause (with which the form must agree), and an indicator feature for lemmas that build synthetic future forms.[11] We trained the classifier on the PDT 2.0 training set, using features from automatic t-tree analysis, with target classes computed from gold-standard annotation projected through alignment. Our setup yields a target structure accuracy of 96.98% on the PDT 2.0 development test set.[12] The errors mostly involve rare combinations of multiple modal verbs, archaic expressions, or errors in the alignment of automatic and gold-standard annotation.

The compound verb form generation module works well for the purpose of t-tree to surface dependency tree conversion, producing more accurate dependency structures than the rule-based module from the Treex/TectoMT Czech surface realizer and matching its output in terms of linear word forms.[13]

## 4.4 Statistical Morphology Generation

In (Dušek and Jurčíček, 2013), we applied statistical methods to address the problem of word inflection generation for our surface realizer, i.e., deducing the correct inflected word form given its lemma (base form) and the desired morphological properties (see Figure 4.4; it corresponds to Step 10 of the pipeline described in Section 4.1). This problem is often ignored altogether in surface realization as NLG systems are very often applied to languages with little morphology, such as English, where a small set of handwritten rules or the direct use of word forms in the symbolic representation or templates is usually sufficient, and so the main focus of these systems lies on syntax and word order.

[10] See Hajič (2004) for more details on the part-of-speech and morphological tags in PDT.

[11] There are only a handful of such verbs in Czech; most verbs use an analytical future form.

[12] The development set has not been used to tune the classifier.

[13] The internal dependency structure of verbal groups is not exploited in the rest of realizer pipeline; therefore, dependency structure accuracy is not an advantage in our application.

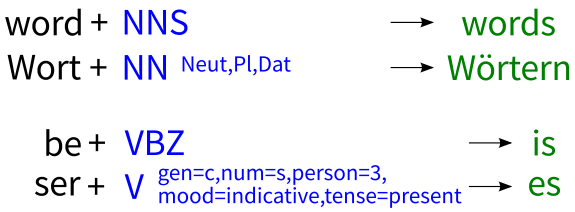


Figure 4.4: The task of morphological generation is to create fully inflected form (right) from base word form and morphological information (left).

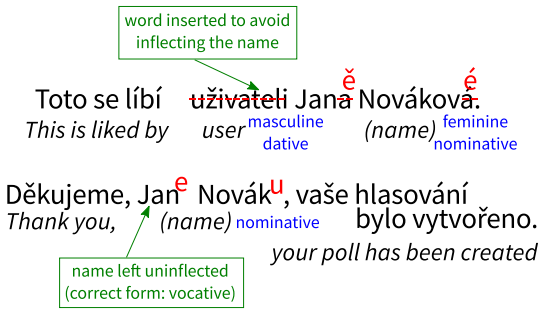


Figure 4.5: Examples of unnatural language resulting from filling in values into templates without any inflection.

The sentences come from Czech translations of Facebook (top) and Doodle (bottom), which both use simple templates to generate personalized texts. Corrections to make the text fluent are shown in red.

There are three traditional approaches to solving the problem of morphological generation: avoiding inflection altogether, rule-based methods, and dictionary based methods (cf. also related work below). While some template-based NLG systems manage to avoid inflection altogether and ensure that a word will keep its base form at all times, this often leads to very unnatural results, especially for languages requiring a lot of inflection in nouns (see Figure 4.5). Using a set of rules covers the problem to a great extent for languages with little morphology such as English (Minnen et al., 2001); however, it can become overly complicated complicated in languages with a complex nominal case system or multiple synthetic verbal inflection patterns, such as Czech or German. Dictionary-based methods (Hajič, 2004; Ptáček and Žabokrtský, 2006) can get very far even for morphology-rich languages, but will not generalize to previously unseen word forms.

To avoid the problems mentioned above, we rely on a statistical approach that learns to predict morphological inflection from annotated data. As a result,

| Lemma | Form | Edit Script | Language |
|---|---|---|---|
| do | doing | >0-ing | *English* |
| llegar | llegó | >2-ó | *Spanish* |
| Mann | Männer | >0-er,3:1-ä | *German* |
| jenž | jež | >2:1- | *Czech* |
| mantenir | mantindran | >0-an,2:1-d,4:1-i | *Catalan* |
| sparen | gespart | >2-t,<ge | *German* |
| vědět | nevíme | >4-íme,<ne | *Czech* |
| be | is | *is | *English* |

Table 4.2: Example edit scripts generated by our system.

The changes are separated by commas. ">" denotes a change at the end of the word, "$N$:" denotes a change at the $N$-th character from the end. The number of deleted characters and their replacement follows in both cases. "<" marks additions to the beginning of a word (regardless of its length). "*" marks irregular forms where the whole word is replaced.

our solution, dubbed *Flect*, manages to produce natural inflection and is more flexible and robust than rules or dictionaries, i.e., and easily trainable for different languages and capable of generalizing to unseen inputs. In the following, we first present our approach, then give an account of our experiments on six different languages, compare our solution to related works, and finally report on the application of morphological generation in our surface realizer.

## Our solution

Similarly to Bohnet et al. (2010) and Durrett and DeNero (2013), we reformulate the task of finding the correct word form as the traditional multiclass classification problem. Instead of finding the desired word form directly (which would induce an explosion of possible target classes), the classifier is trained to find the correct inflection pattern: lemma-form edit scripts – rules describing how to transform the base form into the inflected form – are used as the target classes for our classifier (see Table 4.2). The string distance algorithm of Levenshtein (1966) is adapted to produce diffs on characters, i.e., mappings from lemmas to the target word form that indicate which characters were added, replaced, or removed. As most morphology changes appear at the end of the word, the positions of the changes are indicated from the end of the base form. This approach is similar to the way the morphological analyzers of Hajič (2004) and Chrupała et al. (2008) find the lemma for an unknown word, and it also

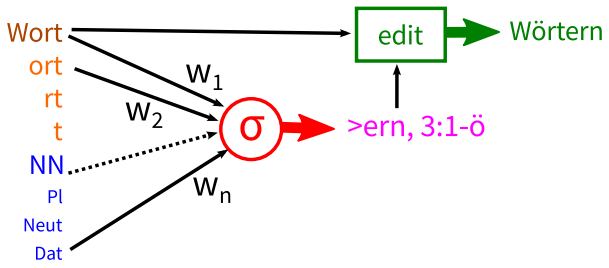


Figure 4.6: Inflection prediction workflow (model predicting prefixes is omitted).

resembles the morphology generation technique used by Bohnet et al. (2010). In Flect, we have introduced several enhancements to this basic scenario:

- changes at the beginning of a word are treated separately as they are typically independent of the word length;
- completely irregular forms are treated as separate classes (this typically concerns a small number of very frequent words, e.g., English *be* → *am*, *is*, *are*);
- casing is disregarded in the edit scripts.

We used the LIBLINEAR logistic regression classifier[14] of Fan et al. (2008) with the following feature types: lemma, part-of-speech tag, other morphological features, and suffixes of the lemma of up to 4 characters. The last feature type allows the classifier to generalize to unknown lemmas since inflection depends mostly on suffixes in many languages.

In addition, we use combinations of morphological features, and certain properties of adjacent words for some languages.[15] We trained separate models for changes at the beginning and at the end of the word, since these two phenomena are often orthogonal (e.g. the usages of the Czech negation prefix *ne-* or the German infinitive prefix *zu-* are quite self-contained phenomena).

Our word inflection prediction schema looks as follows:

1. Using the classifier, predict an edit script for changes at the end or in the middle of the word, or an irregular form.

[14] See Section 1.4.

[15] This is used to model congruence in some languages where it is not described by the morphological tag (e.g., *is* vs. *are* in English) as well as phonetic changes due to surrounding words (e.g., the article *l'* vs. *el* in Catalan). Features specific to each language are described in Table 4.3.

| Language | Additional features | Rule (%) accuracy | Form accuracy (%) Total | -Punc | InflF | UnkF |
|---|---|---|---|---|---|---|
| English | W-1/LT | 99.56 | 99.56 | 99.49 | 97.76 | 98.26 |
| German | W-1/LT, MC | 96.66 / 99.91 | 96.46 | 98.01 | 92.64 | 89.63 |
| Spanish | MC | 99.05 / 99.98 | 99.01 | 98.86 | 97.10 | 91.11 |
| Catalan | W+1/C1, MC | 98.91 / 99.86 | 98.72 | 98.53 | 96.49 | 94.24 |
| Japanese | MC | 99.94 / 100.00 | 99.94 | 99.93 | 99.59 | 99.54 |
| Czech | MC | 99.45 / 99.99 | 99.45 | 99.35 | 98.81 | 95.93 |

Table 4.3: Morphology generation results on CoNLL 2009 datasets.

*Model parameters*: Termination criterion is 0.001 for all languages. Regularization cost is 10 for English, German, and Catalan and 100 for the remaining languages. There are no prefix changes in English and therefore, only a single model is applied.

*Additional features*: *MC* = combination of morphological features (German and Japanese: all nonempty subsets, Spanish and Catalan: ditto except sub-part-of-speech, Czech: only case, number, and gender); *W-1/LT* = lemma and part-of-speech tag of the previous word; *W+1/C1* = first character of the following word.

*Rule (edit script) accuracy* is given for the prediction of changes at the end or in the middle and at the beginning of the word, respectively.

The *form accuracy* field shows the percentage of correctly predicted (lowercased) target word forms: *Total* = on the whole evaluation set, *InflF* = only forms that differ from the lemma (i.e. have a non-empty edit script), *UnkF* = forms unseen in the training set.

2. Predict an edit script for the possible addition of a prefix using a separate model.

3. Apply the edit scripts predicted by the previous steps as rules to generate the final inflected word form.

A simplified depiction of the schema is given in Figure 4.6.

## Evaluation

We have tested our Flect morphology generation model on six languages – Catalan, Czech, English, German, Japanese, and Spanish –, using the CoNLL 2009 Shared Task data sets (Hajič et al., 2009) with gold-standard morphology annotation as training corpora. We used grid search on the development sets to find the best-performing classifier parameters.[16] The final performance of

[16]We used primal form classifier setting, L1-norm regularization, and searched for the best termination criterion and regularization strength for the model predicting changes at the end

| Train data part (%) | Czech Accuracy Base | Czech Accuracy Flect | Czech Error reduct. | English Accuracy Base | English Accuracy Flect | English Error reduct. |
|---|---|---|---|---|---|---|
| 0.1 | 62.00 | 76.92 | 39.27 | 89.18 | 95.02 | 53.91 |
| 0.5 | 66.78 | 88.73 | 66.08 | 91.34 | 97.89 | 75.64 |
| 1 | 69.43 | 92.23 | 74.60 | 92.76 | 98.28 | 76.19 |
| 5 | 77.29 | 96.63 | 85.17 | 96.21 | 99.05 | 74.96 |
| 10 | 80.97 | 97.83 | 88.61 | 97.31 | 99.34 | 75.44 |
| 20 | 85.69 | 98.72 | 91.02 | 98.09 | 99.46 | 71.65 |
| 30 | 87.92 | 98.95 | 91.34 | 98.40 | 99.48 | 67.75 |
| 50 | 90.34 | 99.20 | 91.69 | 98.69 | 99.54 | 64.81 |
| 75 | 91.91 | 99.34 | 91.89 | 98.86 | 99.55 | 60.61 |
| 100 | 92.88 | 99.45 | 92.21 | 98.94 | 99.56 | 58.85 |

Table 4.4: Comparison of Flect with a dictionary baseline on different training data sizes.

All numbers are percentages. We include the accuracy of the dictionary baseline (*Base*) and our method (*Flect*), along with the relative error reduction.

our classifiers on the evaluation test sets, along with parameter values and language-specific settings, is given in Table 4.3.

One can see that the system is able to predict the majority of word forms correctly and performs well even on data unseen in the training set. When manually inspecting the errors produced by the system, we observed that in some cases the system in fact assigned a form synonymous to the one actually occurring in the test set, such as *not* instead of *n't* in English or *také* instead of *taky* (both meaning *also*) in Czech. However, most errors are caused by the selection of a more frequent rule, even if incorrect given the actual morphological features. The lower score for German is caused partly by the lack of syntactic features for the highly ambiguous adjective inflection and partly by a somewhat problematic lemmatization of punctuation.[17]

In a specific experiment targeting English and Czech, we compared Flect with a baseline that uses a dictionary collected from the same data and leaves unseen base word forms intact. The results are shown in Table 4.4 for different sizes of the training data. We can see that our approach is capable of reaching

or in the middle of the word. As the changes at the beginning of the word are much simpler, changing parameters for this model did not have a significant impact on performance. We thus used the same parameters for both models.

[17] All punctuation has the same lemma "_", and the part-of-speech tags do not distinguish unambiguously.

high performance even with relatively small amounts of training instances. Our approach maintains a significantly[18] higher accuracy when compared to the baseline for all training data sizes. The overall performance difference becomes smaller as the training data grow; however, the relative error reduction shows a different trend: the improvement stays stable, decreasing slightly for English where unknown word forms are more likely to be base forms of unknown lemmas, but increasing steadily for Czech where unknown word forms are more likely to require inflection.

Though the number of unseen word forms keeps declining with increasing amounts of training data, which plays in favor of the dictionary method, unseen inputs will always occur and may become very frequent for out-of-domain data. Our Flect morphology generation system is therefore beneficial – at least as a backoff for unseen forms – even if a large-coverage morphological dictionary is available. Flect maintains a somewhat smaller performance margin of 1.2% accuracy absolute even over Hajič (2004)'s morphological generator based on a very large dictionary, which reaches 98.25% accuracy on the Czech evaluation set.

## Related Work in Morphology Generation

There have been few previous attempts at statistical morphology generation in the context of surface realization. NLG systems either use word forms directly without a morphological generation step (e.g., Angeli et al., 2010; Mairesse et al., 2010), apply hand-built lexicons and rules (e.g., Lavoie and Rambow, 1997; Ptáček and Žabokrtský, 2006; Gatt and Reiter, 2009; de Kok, 2013). If applied, morphological generation in the context of MT tends to use dictionaries – hand-built (Žabokrtský et al., 2008), learned from data (Toutanova et al., 2008; Fraser, 2009; El Kholy and Habash, 2012), or a combination thereof (Popel and Žabokrtský, 2009).

The only previous morphological generator in surface realization known to us is by Bohnet et al. (2010). However, their morphology generation was only a component of a complex generation system. Their system is very similar to ours, but their method did not attempt to generalize beyond seen inputs, and they did not perform any deep analysis of the capabilities of their methods. We propose several improvements and provide a detailed evaluation of a statistical morphological inflection system, including more languages into the evaluation and focusing on robustness to unseen inputs.

[18] Significance at the 99% level has been assessed using paired bootstrap resampling (Koehn, 2004).

Also related to our problem are morphology models developed in a standalone setting which focus on discovering complete morphological paradigms[19] from data. Dreyer et al. (2008) define joint distributions of string pairs as a log-linear model, connecting the base form with an inflected form. They apply weighted finite-state transducers over characters for training and inference. Separate transducers must be trained for individual grammatical categories and the model can be computationally expensive, summing over all possible transductions from the base form into the inflected form. Dreyer and Eisner (2009) combine multiple inflected forms and model full inflectional paradigms as a product of pairwise distributions in the form of a Markov random field. They train their models on dictionary inflection tables, disregarding frequency of the individual forms. Dreyer and Eisner (2011) then include frequency information from a plain text or part-of-speech tagged corpus, using Gibbs sampling for learning. Durrett and DeNero (2013) use a CRF classifier (Lafferty et al., 2001) trained on morphological dictionaries to predict morphological changes at all positions of a word. They perform a joint classification for the whole morphological paradigm: The target classes are rules representing a list of changes for all possible forms in the paradigm.

As opposed to our work, Dreyer et al.'s models allow even unobserved morphological transformations to be produced. However, this rarely happens in practice (unknown words tend to follow the patterns set by more frequent words) and their models are computationally expensive as a result, requiring approximate inference to reach the solution. The model of Durrett and DeNero (2013) is similar to ours in terms of allowed morphological transformations and efficiency. However, both the model of Durrett and DeNero (2013) and the works of Dreyer et al. only work with dictionaries, require full inflection tables for training, and do not take occurrence frequency of the individual morphological changes into account.[20] Our approach does not require any dictionary, a morphologically annotated training corpus is sufficient[21] and partially unobserved inflection paradigms pose no problem.

[19] A morphological inflection paradigm is the set of all inflected forms derived from the same base form (lemma), e.g., a noun declension in all possible cases for the given language. Morphological dictionaries represent inflection paradigms in inflection tables.

[20] With the exception of Dreyer and Eisner (2011), which uses both training inflection tables and a plain text corpus.

[21] In practice, morphologically annotated corpora for many languages are freely available (Nivre et al., 2016). The availability of morphological dictionaries is, to our knowledge, more limited. Wiktionary (`https://en.wiktionary.org/`, Accessed: October 11, 2016) was used successfully by Durrett and DeNero (2013) despite its rather limited coverage, which varies widely by language.

Multiple papers in morphology generation have been published after we performed our experiments in (Dušek and Jurčíček, 2013), and the problem has been highlighted by the 2016 SIGMORPHON shared task (Cotterell et al., 2016), with specially created datasets for 10 languages and nine competing systems mostly building on top of previous approaches.

Some works continue the attempts to reconstruct whole morphological inflection paradigms based on dictionaries. Sylak-Glassman et al. (2015) extend the work of Durrett and DeNero (2013) by mining consistent word inflection tables from Wiktionary for over 80 languages. Ahlberg et al. (2014) extract paradigms from inflection tables using longest common subsequence methods, then apply heuristics based on frequency to select the best paradigm for a given base form. Ahlberg et al. (2015) then replace the heuristics with an SVM classifier.

In the context of MT, Nicolai et al. (2015) solve the same task as ours: generating single inflected forms based on lemma and morphological categories. They concatenate the base form with the desired morphological properties and apply a discriminative string transducer based on a semi-Markov model, which uses character $n$-gram features from both source and target, to arrive at the inflected form. Recently, Faruqui et al. (2016) tackled the problem of generating single inflected forms using RNNs. They build one network per inflection form (e.g., "dative plural in nouns"), where an RNN encoder over characters encodes the base form into a fixed-length vector and an RNN decoder decodes the characters of the inflected form. In addition, they apply a language model for beam-search decoding. Kann and Schütze (2016), the winning approach of the SIGMORPHON shared task, also uses encoder-decoder RNNs to generate inflection. They encode the characters of the base form along with morphological properties of the base and inflected forms, and the decoder produces the inflected form character-by-character. They use an attention model (Bahdanau et al., 2015, see Section 6.2) for decoding and add a simple correction model based on minimum edit distances.

## Application in the Surface Realizer

As already noted in Section 4.1, Flect has been integrated into our English surface realizer, which is also used in the TectoMT translation system (see Section 4.2).[22] Flect has also been employed in the Basque and Spanish surface

[22]For this purpose, the model has been retrained on PCEDT 2.0 (Hajič et al., 2012) Sections 02-21 using noisy lemma and morphology information resulting from the analysis and synthesis round-trip (see Section 4.1) as inputs. Original word forms projected through monolingual word alignment (Rosa et al., 2012) were used as targets to build inflection edit scripts.

| Variant | BLEU (%) |
|---|---|
| Baseline (MorphoDiTa) | 73.55 |
| Flect alone | 77.04 |
| MorphoDiTa + Flect as a backoff | 77.47 |

Table 4.5: Evaluating morphology generation within the surface realizer.

Evaluation on PCEDT 2.0 Sections 22 and 23. Round-trip: analyzed and regenerated sentences are compared to the original ones.

realizers built for the TectoMT system within the QTLeap project (Aranberri et al., 2016; Popel et al., 2015b).

To test Flect's performance extrinsically, we used the same English deep syntax round-trip scenario as described in Section 4.1 and compare the overall score with and without Flect being applied. We compare the following three setups for morphological generation, with the rest of the pipeline unchanged:

1. A baseline setting, where MorphoDiTa English morphological dictionary (Straková et al., 2014) is used alone. If MorphoDiTa provides multiple options for a combination of a word and a morphological tag, the first option is selected. If the word-tag combination is not in the dictionary, the word is not inflected.

2. Flect being used alone for all words.

3. A combination of MorphoDiTa and Flect. Here, MorphoDiTa output is used only if it provides just one inflected form for a given input lemma-tag combination. If MorphoDiTa returns none or more than one option, Flect is used as a backoff.

Using PCEDT 2.0 (Hajič et al., 2012) Sections 22 and 23, we measured the BLEU score of the analyzed and regenerated sentences against the original sentence as reference. The results are shown in Table 4.5.

We can see that the settings involving Flect are clearly superior to using MorphoDiTa alone, with a 3.5% BLEU gap. Combining the MorphoDiTa dictionary with Flect as a backoff yields slightly better results than using Flect alone. A manual examination of a sample of the data confirms the automatic scores: with MorphoDiTa alone, many words remain uninflected. On the other hand, Flect uses wrong inflection in some cases – here, MorphoDiTa's dictionary can often compensate for that. This confirms Flect's usability in a real-world scenario.

## 4.5 Discussion

Building on top of an earlier Czech realizer, we have created a surface realizer from t-trees for English and evaluated it successfully in a round-trip test as well as in a transfer-based MT system; we showed that the realizer is able to produce valid English strings for a broad domain. This realizer will be used in Chapters 5 and 6 to postprocess the output of sentence planning within our dialogue NLG setting.

The realizer is mostly rule-based, yet conceptually very simple and easy to build. We have also shown two experiments that introduce statistical components to surface realization from t-trees. The compound verbal groups generation experiment described in Section 4.3 was successful but remains a proof of concept; on the other hand, our Flect morphology generation system (see Section 4.4) has been put into practice.

Flect has been shown to be very simple and efficient; it is now used in our t-tree realizer for English as well as in Treex/TectoMT realizers for Spanish and Basque. The approach we developed in 2013 was state-of-the-art, but has since been superseded by RNN-based solutions that do not require any hand-built features and are thus easier to train; nevertheless, Flect still remains useful.

Given today's RNN-based approaches to many problems and also our results with joint generation in Chapter 6, we believe that the whole problem of surface realization could now be solved using RNNs, but we leave this experiment for future work.

Our English realizer is freely available[23] for download as part of the Treex/TectoMT NLP framework under:

`https://github.com/ufal/treex`

The realizer can be triggered in Treex using the `Scen::Synthesis::EN` scenario. Flect, our morphology generation system, is also freely available[24] for download on GitHub, here:

`https://github.com/UFAL-DSG/flect`

[23] Treex is distributed under the GNU and Artistic licenses, same as the Perl programming language used for its implementation.

[24] Flect uses the Apache 2.0 free license.

# 5

# Perceptron-based Sentence Planning

In this chapter, which is partially based on (Dušek and Jurčíček, 2015), we present a our first experiments with a novel approach to NLG for SDSs that does not require fine-grained alignment in training data. This approach follows the NLG pipeline division into sentence planning and surface realization (see Sections 2.1 and 3.4) and uses t-trees for sentence plans (see Section 3.5). We focus on the sentence planning approach based on A*-search and perceptron ranking in the text of this chapter, and we use the surface realizer described in Chapter 4 in our experiments.

The A*/perceptron approach described here has since been superseded by a newer NN-based generator (see Chapter 6), but it advanced the state-of-the art as the first approach allowing simpler training than previous setups where fine-grained alignments were required for training (see Section 3.2) – our sentence planner includes alignment learning directly into the training process. In addition, unlike most previous approaches to trainable sentence planning (e.g., Walker et al., 2001a; Stent et al., 2004), our system does not require a handcrafted base module.

This chapter is divided as follows: We first give a high-level description of the overall architecture of the whole generation setup (both sentence planning and surface realization) in Section 5.1. The following three sections then focus on the sentence planning approach, which is the main subject of this chapter. An overview over the whole sentence planning algorithm is given in Section 5.2; its two main components, candidate generator and scorer, are then explained in more detail in Sections 5.3 and 5.4, respectively.

The following three sections deal with our experiments on the BAGEL restaurant recommendation data set of Mairesse et al. (2010). Section 5.5 gives an account of the full experimental setup we used to evaluate our approach, Section 5.6 then presents our mostly encouraging results on the BAGEL set. In Section 5.7, we describe problems encountered in our follow-up experiments with the A*/perceptron generator, which finally led us to consider a different generator architecture (see Chapter 6).

The final Section 5.8 compares our generator to previous approaches, summarizes our results, and closes with an outlook into the following Chapters 6, 7, and 8.

## 5.1 Overall Generator Architecture

The overall schema of the whole generation procedure is depicted in Figure 5.1. In the first stage, a statistical sentence planner generates t-trees (deep-syntactic dependency trees) from the input meaning representation. These are converted into plain text sentences in the second stage by the (mostly rule-based) surface realizer.

The general pros and cons of taking the two-step approach to generation have been described in Sections 2.1 and 3.4. Our decision to only use separate sentence planner and surface realizer when experimenting with A*-search-based generation with perceptron ranking was motivated mainly by simplifying the task – the planner does not need to handle surface morphology and auxiliary words. Since the hypothesis space grows very fast with the number of output tree nodes and the number of different node labels, this helps to limit the number of explored candidate outputs.

### Data Formats

For our experiments throughout this chapter, we use DAs in our triplet list representation (DA type–slot–value) described in Section 3.1 as the input to our generator. We convert the DAs of the BAGEL dataset (Mairesse et al., 2010) into this format; these originally consist of the DA type, which is always *inform*,[1] and a list of slot-value pairs (SVPs) that contain information about a restaurant, such as food type or location (see the examples in Figure 3.2 and on the top left of Figure 5.1). Our generator can be easily adapted to a different input MR format

[1] No other DA types occur in the BAGEL set, but our generator is prepared to handle different DA types.

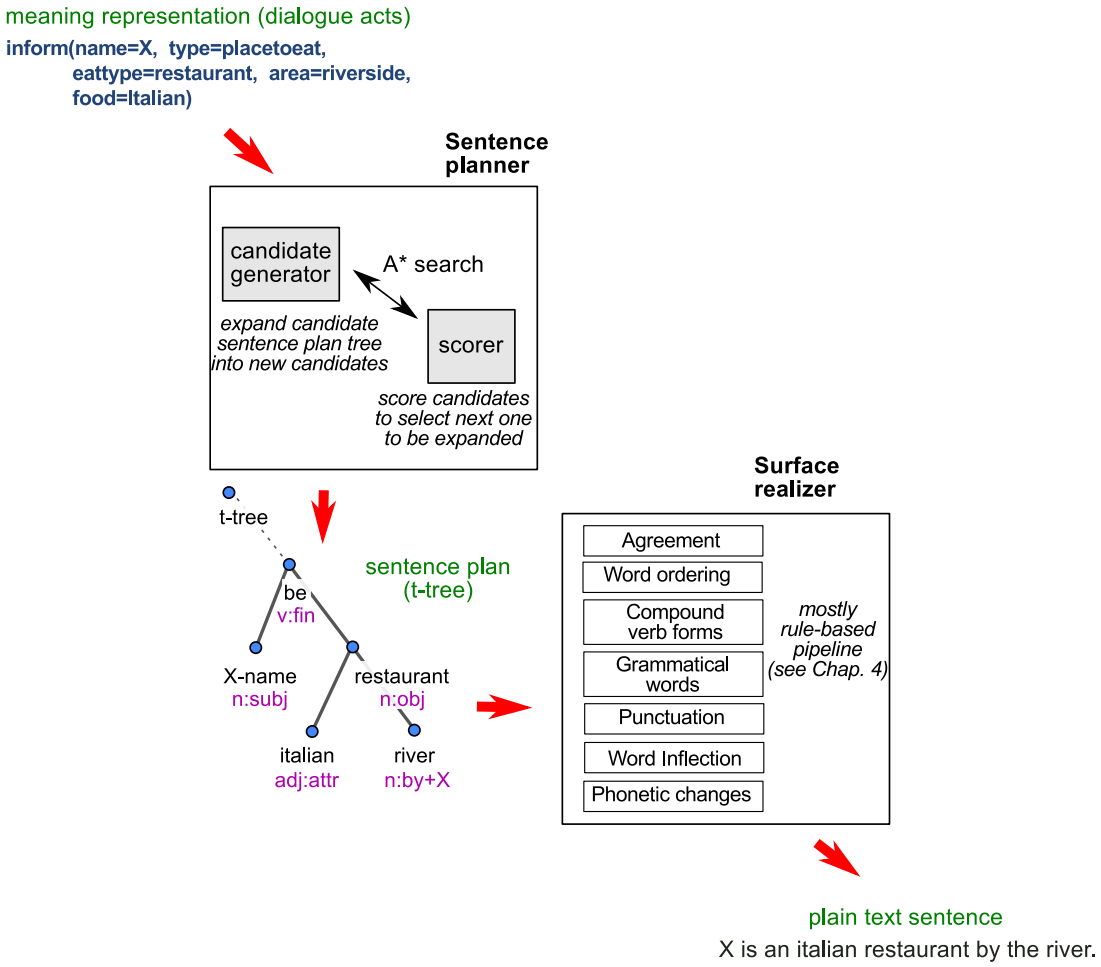


Figure 5.1: Overall structure of our generator.

– it is sufficient to adapt some of the candidate generator rules (see Section 5.3) and the feature set (see Section 5.5).

The sentence plan format used in our experiments in our chapter is the simplified t-tree deep syntactic formalism as described in Section 3.5 (see also Figure 5.1 in the middle) – dependency trees in surface word order, with two attributes per node: lemma (t-lemma or deep lemma, representing the concept)[2] and formeme (morphosyntactic tag). Deep-syntax annotation of sentences in the training set is needed to train the sentence planner, but we assume automatic annotation and reuse an existing deep-syntactic analyzer from the Treex NLP framework (see Section 3.4).

[2]For brevity, we will use the term "lemma" to refer to t-lemmas in this chapter as there is no risk of confusion with surface lemmas.

### Surface Realizer

The experiments in this chapter concentrate on sentence planning, but note that the output of our sentence planner is postprocessed by the surface realizer described in Chapter 4.

As our evaluation is done on the plain text outputs of the surface realizer (see Section 5.5), we tested the suitability of our surface realizer for our domain in a simple round-trip test, similarly to standalone realizer tests performed in Sections 4.2 and 4.4. We first used automatic analysis from the Treex toolkit, and then we generated the resulting deep syntax trees back to sentences. On the whole BAGEL data set, this round-trip achieved a BLEU score of 89.79% against the original sentences (single reference; see Section 3.6), showing only minor differences between the input sentence and generation output, mostly in punctuation.

## 5.2 Sentence Planner Architecture

The sentence planner is based on a variant of the A* algorithm (Hart et al., 1968; Och et al., 2001; Koehn et al., 2003). It starts from an empty sentence plan tree and tries to find a path to the complete, optimal sentence plan by iteratively adding nodes to the currently "most promising" incomplete sentence plan. It uses the following two subcomponents to guide the search:

- a *candidate generator* that is able to incrementally generate candidate sentence plan trees (expanding an incomplete sentence plan; see Section 5.3),
- a *scorer/ranker* that scores the appropriateness of the sentence plan trees for the input DA (and selects the next sentence plan tree to be expanded; see Section 5.4).

As noted in Section 5.1, the sentence planner is trained using pairs of input DAs and the corresponding sentence plan trees, which can be obtained by automatic analysis in the Treex NLP toolkit (cf. Section 3.4).

### Sentence Planning Algorithm

During the search process, the sentence planner algorithm keeps two sets of hypotheses, i.e., candidate sentence plan trees, sorted by their score – hypotheses to expand (*open set*) and hypotheses already expanded (*closed set*). Its basic workflow can be described with the following initialization setting, main loop, and stopping criterion:

*Init:* Start from an *open set* with a single empty sentence plan tree and an empty *closed set*.

*Loop:*
1. Select the best-scoring candidate $c$ from the *open set*. Add $c$ to *closed set*.
2. Given $c$, the candidate generator generates **C**, a set of possible "successors" to $c$. These are trees that have more nodes than $c$ and are deemed viable. Note that **C** may be empty.
3. The scorer scores all the successors from **C** and if they are not already in the *closed set*, it adds them to the *open set*.
4. Check (for the stopping criterion) if the *closed set* is non-empty and if the best successor in the *open set* has a better score than the best candidate in the *closed set*.

*Stop:* The algorithm finishes if the top score in the *open set* is lower than the top score in the *closed set* for $s$ consecutive iterations, or if there are no more candidates in the *open set*. It returns the best-scoring candidate over both sets.

The number of non-improving consecutive iterations $s$ is a free parameter.

## 5.3 Generating Sentence Plan Candidates

Given a sentence plan tree, the candidate generator generates its successors by adding one new node in all possible positions and with all possible lemmas and formemes (see Figure 5.2). The input sentence plan is incomplete in a typical case and may be even empty (technical root only), which is the case at the start of the search process.

While a naive implementation – trying out any combination of lemmas and formemes found in the training data in any possible positions – works in principle, it leads to an unmanageable number of candidate trees even for a very small domain and a very small number of nodes.[3] Therefore, we include several rules that limit the number of trees generated:

1. *Lemma-formeme compatibility* – only nodes with a combination of lemma and formeme seen in the training data are generated.

[3]Even disregarding different tree topologies, the number is exponential to the number of possible different nodes (i.e., number of lemmas $\times$ number of formemes). For the very small BAGEL domain, there are 129 different lemmas and 46 different formemes in the whole data set. Therefore, the number of possible is trees of size $k$ is at least $(129 \cdot 46)^k$. This yields around 200 trillion for $k = 3$. Due to different tree topologies, this number is even higher in practice.

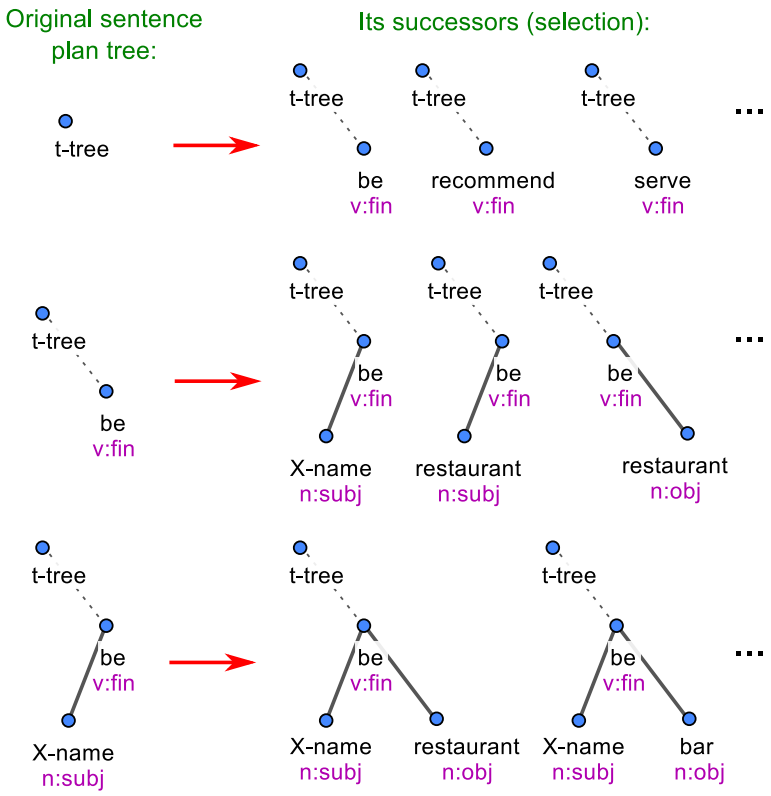


Figure 5.2: Candidate generator example inputs and outputs.

2. *Syntactic viability* – the new node must be compatible with its parent node (i.e., this parent-child pair, including the dependency left/right direction, must be seen in the training data).

3. *Number of children* – no node can have more children than the maximum for this lemma-formeme combination seen in the training data.

4. *Tree size* – the generated tree cannot have more nodes than trees seen in the training data. The same limitation applies to the individual depth levels – the training data limit the number of nodes on the $k$-th depth level as well as the maximum depth of any tree.

   This is further conditioned on the input SVPs – the maximums are only taken over training examples that contain at least one of the SVPs that appear on the current input.

5. *Weak semantic compatibility* – we only include nodes that appear in the training data alongside the elements of the input DA, i.e., nodes that appear in training examples containing SVPs from the current input.

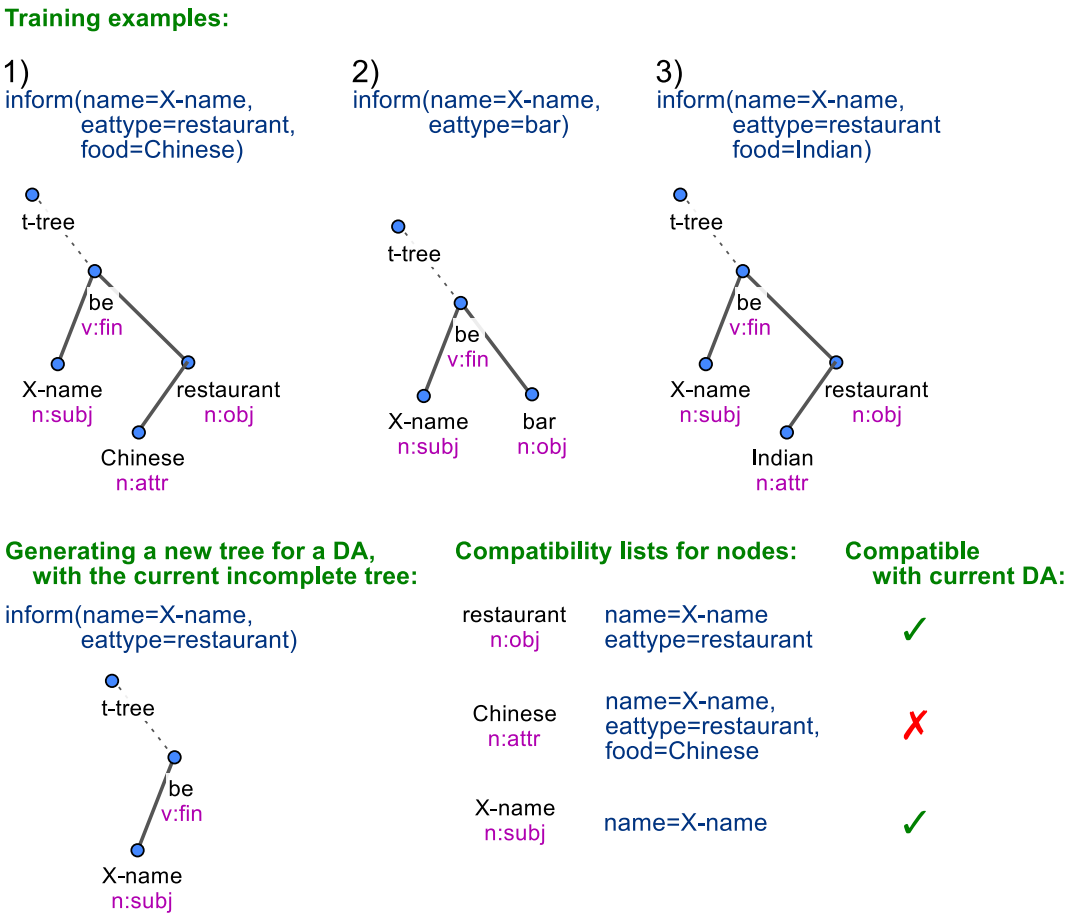


Figure 5.3: An example of the strong semantic compatibility rule in the candidate generator (Rule 6).

For each possible node to be generated, the compatibility lists are made using an intersection of SVPs occurring in DAs that appear alongside this node in the training data.

For example, the node restaurant/n:obj appears in training examples 1 and 3. If we take the intersection of SVPs in DAs 1 and 3, we get name=X-name, eattype=restaurant. Both SVPs are included in the current DA to be generated, hence the node restaurant/n:obj is compatible with the current DA.

On the other hand, the node Chinese/n:attr only appears in training example 1. Therefore, the intersection of SVPs in training examples containing this node is equal to exactly all SVPs of training example 1. Since the DA to be generated is not a superset of the DA in training example 1, the node Chinese/n:attr is not compatible with the DA according to this rule.

6. *Strong semantic compatibility* – for each node (lemma and formeme), we make a "compatibility list" of SVPs and slots that are present in all training data examples containing this node. We then only allow generating this node if all of them are present in the current input DA (see Figure 5.3 for an example).

   To allow for more generalization, this rule can be applied just to lemmas (disregarding formemes), and a certain number of SVPs/slots from the compatibility list may be required at maximum (see Section 5.5 for the application of this).

Only Rules 4 (partly), 5, and 6 depend on the format of the input meaning representation. Using a different MR than DAs would require changing these rules to work with atomic substructures of the new MR instead of SVPs.

While especially Rules 5 and 6 exclude a vast number of potential candidate trees, this limitation is still much weaker than using hard alignment links between the elements of the MR and the output words or phrases. It leaves enough room to generate many combinations unseen in the training data (cf. Section 5.6) while keeping the search space manageable. To limit the space of potential tree candidates even further, one could also use automatic alignment scores between the elements of the input MR and the tree nodes obtained using a tool such as GIZA++ (Och and Ney, 2003).

## 5.4 Scoring Sentence Plan Trees

The scorer for the sentence plan tree candidates produced by the candidate generator (see Section 5.3) is a function that maps global features from the whole sentence plan candidate tree $t$ and the input DA $d$ to a real-valued score that describes the fitness of $t$ in the context of $d$.[4]

We first describe the basic version of the scorer and then our two improvements – differing subtree updates and future promise estimation.

### Basic perceptron scorer

The basic scorer is based on the linear perceptron ranker of Collins and Duffy (2002), where the score is computed as a simple dot product of the features and the corresponding weight vector:

$$\text{score}(t, d) = \mathbf{w}^\top \cdot \text{feat}(t, d) \tag{5.1}$$

[4]Note that this scoring algorithm is applicable to any MR type, not just DAs, given an appropriate set of features (cf. also Section 5.5).

In the training phase, the weights **w** are initialized with a value of 1. For each input DA, the system tries to generate the best sentence plan tree given current weights, $t_{top}$. The score of this tree is then compared to the score of the correct gold-standard tree $t_{gold}$.[5] If $t_{top} \neq t_{gold}$ and the gold-standard tree ranks worse than the generated one (score$(t_{top}, d) >$ score$(t_{gold}, d)$), the weight vector is updated by the feature value difference of the generated and the gold-standard tree:

$$\mathbf{w} = \mathbf{w} + \alpha \cdot (\text{feat}(t_{gold}, d) - \text{feat}(t_{top}, d)) \tag{5.2}$$

In (5.2), $\alpha$ is a predefined learning rate.

## Differing subtree updates

In the basic version described above, the scorer is trained to score full sentence plan trees. However, it is also used to score incomplete sentence plans during the decoding. This leads to a bias towards bigger trees regardless of their fitness for the input DA. Therefore, we introduced a novel modification of the perceptron updates to improve scoring of incomplete sentence plans: In addition to updating the weights using the top-scoring candidate $t_{top}$ and the gold-standard tree $t_{gold}$ (see above), we also use their *differing subtrees* $t^i_{top}, t^i_{gold}$ for additional updates.

Starting from the common subtree $t_c$ of $t_{top}$ and $t_{gold}$, pairs of differing subtrees $t^i_{top}, t^i_{gold}$ are created by gradually adding nodes from $t_{top}$ into $t^i_{top}$ and from $t_{gold}$ into $t^i_{gold}$ (see Figure 5.4). To maintain the symmetry of the updates in cases where the sizes of $t_{top}$ and $t_{gold}$ differ, more than one node may be added in one step.[6] The additional updates then look as follows:

$$\begin{aligned}
&t^0_{top} = t^0_{gold} = t_c \\
&\text{for } i \text{ in } 1, \ldots \min\{|t_{top}| - |t_c|, |t_{gold}| - |t_c|\} - 1 : \\
&\quad t^i_{top} = t^{i-1}_{top} + \text{node(s) from } t_{top} \\
&\quad t^i_{gold} = t^{i-1}_{gold} + \text{node(s) from } t_{gold} \\
&\quad \mathbf{w} = \mathbf{w} + \alpha \cdot (\text{feat}(t^i_{gold}, d) - \text{feat}(t^i_{top}, d))
\end{aligned}$$

[5] Note that the "gold-standard" sentence plan trees are actually produced by automatic annotation (see Section 5.1). For the purposes of scoring, they are, however, treated as gold standard.

[6] For example, if $t_{gold}$ has 6 more nodes than $t_c$ and $t_{top}$ has 4 more, there will be 3 pairs of differing subtrees, with $t^i_{gold}$ having 2, 4, and 5 more nodes than $t_c$ and $t^i_{top}$ having 1, 2, and 3 more nodes than $t_c$.

We have also evaluated a variant where both sets of subtrees $t^i_{gold}, t^i_{top}$ were not equal in size, but this resulted in degraded performance.

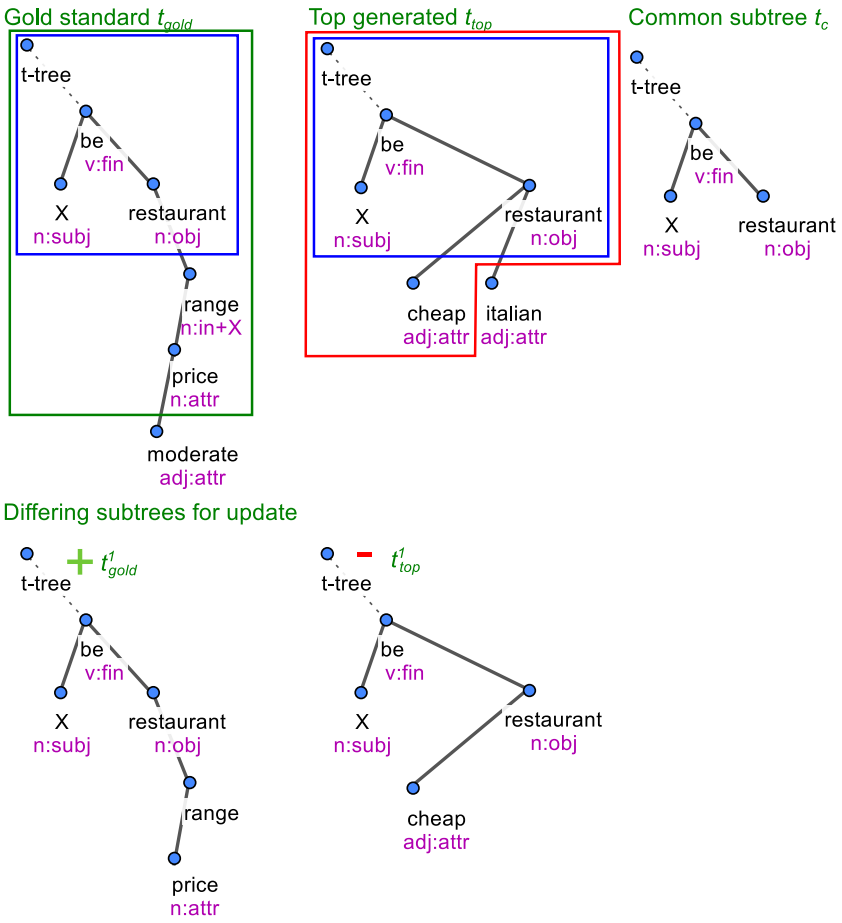


Figure 5.4: An example of differing subtrees.

The gold standard tree $t_{gold}$ has three more nodes than the common subtree $t_c$, while the top generated tree $t_{top}$ has two more. The common subtree $t_c$ is marked with a blue rectangle in $t_{gold}$ and $t_{top}$. Only one pair of differing subtrees $t^1_{gold}, t^1_{top}$ is built, where two nodes are added into $t^1_{gold}$ and one node into $t^1_{top}$. $t^1_{gold}$ is marked with a green rectangle in $t_{gold}$, $t^1_{top}$ with a red rectangle in $t_{top}$.

## Future promise estimation

To further improve scoring of incomplete sentence plan trees, we incorporate a simple *future promise* estimation for the A* search, intended to boost scores of sentence plans that are expected to further grow (see Figure 5.5).[7] It is based on the expected number of children $E_c(n)$ of different node types (lemma-formeme

[7]Note that this is not the same as future path cost in the original A* path search, but it plays an analogous role: weighing hypotheses of different size.

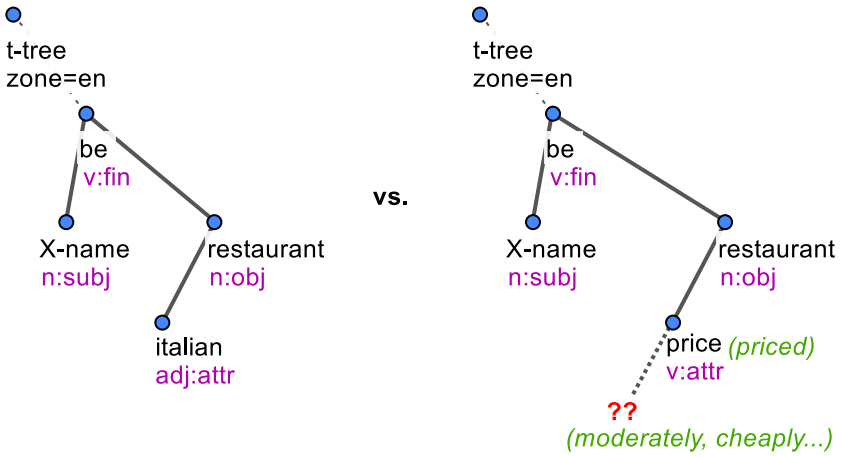


Figure 5.5: Future promise example.

The tree on the left represents a complete sentence. The tree on the right has the same number of nodes, but does not represent a complete sentence. It is expected to grow by at least one node; therefore, its future promise estimate should be higher than for the tree on the left.

pairs).[8] Given all nodes $n_1 \ldots n_{|t|}$ in a sentence plan tree $t$, the future promise of $t$ is computed in the following way:

$$\text{fp}(t) = \lambda \cdot \sum \mathbf{w} \cdot \sum_{i=1}^{|t|} \max\{0, E_c(n_i) - c(n_i)\} \tag{5.3}$$

In (5.3), $c(n_i)$ is the current number of children of node $n_i$, $\lambda$ is a preset weight parameter, and $\sum \mathbf{w}$ is the sum of the current perceptron weights. Multiplying by the weights sum makes future promise values comparable to trees scores.

Future promise is added to tree scores throughout the tree generation process, but it is disregarded for the termination criterion in the *Stop* step of the generation algorithm and in perceptron weight updates.

## Averaging weights and parallel training

To speed up training using parallel processing, we use the iterative parameter mixing approach of McDonald et al. (2010), where training data are split into several parts and weight updates are averaged after each pass through the training data. Following Collins (2002), we record the scorer perceptron weights

[8] $E_c(n)$ is measured as the average number of children over all occurrences of the given node type in the training data. It is expected to be domain-specific.

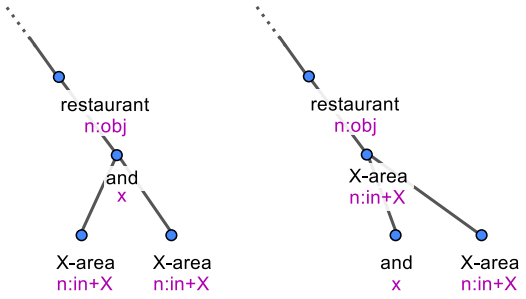


Figure 5.6: Coordination structures conversion: original Treex annotation (left) and our converted format (right).

after each training pass, take an average at the end, and use this as the final weights for prediction.

## 5.5 Experimental Setup

In this section, we describe the data set used to evaluate our sentence planner, as well as the required preprocessing steps, and the settings of our sentence planner that are specific to the data set.

### Dataset

We performed our experiments on the freely available BAGEL data set of Mairesse et al. (2010, see Section 2.4). It contains a total of 404 sentences from a restaurant information domain (describing the restaurant location, food type, etc.), which correspond to 202 dialogue acts, i.e., each DA has two paraphrases. Restaurant names, phone numbers, and other "non-enumerable" properties are delexicalized throughout the generation process (see Section 3.3). Note that while the data set contains alignment of source SVPs to target phrases, we do not use it in our experiments.

For sentence planner training, we automatically annotate all the sentences with the corresponding t-trees using the deep syntactic analyzer from the Treex NLP toolkit (see Section 3.4). The annotation obtained from the Treex analyzer is further simplified for the sentence planner in two ways:

- Only lemmas and formemes are used in the sentence planner; the most frequent values of grammatemes (grammatical attributes) encountered

in the training data for the particular node types are used for the surface realization step (see Section 3.5).

- We convert the dependency tree representation of coordination structures into a format inspired by Universal Dependencies (Nivre et al., 2016, cf. Section 2.4). In the original Treex annotation style, the conjunction heads both conjuncts, whereas in our modification, the first conjunct is at the top, heading the coordination and the second conjunct (see Figure 5.6).

  The coordination structures can be easily converted back for the Treex/TectoMT-style surface realizer, and the change makes the task easier for the sentence planner: it may first generate one node and then decide whether it will add a conjunction and a second conjunct.

## Generator Settings

In our experiments, we use all the limitation heuristics for the candidate generator described in Section 5.3. For strong semantic compatibility (Rule 6), we use just lemmas and require at most 5 SVPs/slots from the lemma's compatibility list in the input DA.

For our sentence plan scorer, we use the following feature types (given an input DA and a candidate sentence plan t-tree):

- current t-tree properties – the t-tree depth, total number of nodes, maximum number of children in a single node, and the number of repeated nodes,
- t-tree and input DA – the number of sentence plan tree nodes per one SVP in the input DA and the number of repeated nodes per repeated SVP,
- t-tree node features – lemma, formeme, and number of children of all nodes in the current sentence plan t-tree, and combinations thereof (lemma + formeme, formeme + number of children, all three),
- input DA features – whole SVPs (slot + value), just slots, and pairs of slots in the DA,
- combinations of nodes and input DA features (lemmas and formemes with SVPs and slots; lemmas, formemes, and number of children with SVPs),
- repeat features – occurrence of repeated lemmas and/or formemes in the current t-tree combined with repeated slots in the input DA,

- dependency features – dependency parent-child pairs in the sentence plan t-tree for lemmas and/or formemes, including and excluding their left-right order,
- sibling features – sibling pairs in the sentence plan tree for lemmas and/or formemes, also combined with SVPs,
- bigram features – pairs of lemmas and/or formemes adjacent in the t-tree's left-right order, also combined with SVPs.

All feature values are normalized to have a mean of 0 and a standard deviation of 1, with normalization coefficients estimated from training data.

The feature set can be adapted for a different MR format than DAs – it only must capture all important parts of the new MR, e.g., for a tree-like MR, the nodes and edges, and possibly combinations thereof.

### Scorer Training Setup

Based on our preliminary experiments, we use the following parameter values to train the scorer feature values: The number of passes over the training data is set to 100. We limit $s$, the number of consecutive iterations after which the A* generation terminates if the best score is not improved (see Section 5.2), to 3 for training and 4 for testing. We use a hard maximum of 200 sentence planner iterations per input DA. The learning rate $\alpha$ is set to 0.1. We use training data parts of 36 or 37 training examples (1/10th of the full training set) in parallel training (see Section 5.4). If future promise is used, its weight $\lambda$ is set to 0.3.

## 5.6 Results

Same as Mairesse et al. (2010), we use 10-fold cross-validation where DAs seen at training time are never used for testing, i.e., both paraphrases or none of them are present in the full training set. We evaluate our generator using the automatic BLEU and NIST scores (Papineni et al., 2002; Doddington, 2002, see Section 3.6) against both reference paraphrases for a given test DA. The results are shown in Table 5.1, both for standard perceptron updates and our improvements – differing subtree updates and future promise estimation (see Section 5.4).

Our generator did not achieve the same performance as that of Mairesse et al. (2010) (ca. 67% BLEU).[9] However, our task is substantially harder since

[9] Mairesse et al. (2010) do not give a precise BLEU score number in their paper, they only show the values in a graph.

| Setup | BLEU for training portion | | | | |
|---|---|---|---|---|---|
| | **10%** | **20%** | **30%** | **50%** | **100%** |
| Basic perceptron | 46.90 | 52.81 | 55.43 | 54.53 | 54.24 |
| + Diff-tree updates | 44.16 | 50.86 | 53.61 | 55.71 | 58.70 |
| + Future promise | 37.25 | 53.57 | 53.80 | 58.15 | 59.89 |
| **Setup** | **NIST for training portion** | | | | |
| | **10%** | **20%** | **30%** | **50%** | **100%** |
| Basic perceptron | 4.295 | 4.652 | 4.669 | 4.758 | 4.643 |
| + Diff-tree updates | 3.846 | 4.406 | 4.532 | 4.674 | 4.876 |
| + Future promise | 3.331 | 4.549 | 4.607 | 5.071 | 5.231 |

Table 5.1: Automatic evaluation on the BAGEL data set (averaged over all ten cross-validation folds)

"Training portion" denotes the percentage of the training data used in the experiment. "Basic perceptron" = basic perceptron updates, "+ Diff-tree updates" = with differing subtree perceptron updates, "+ Future promise" = with future promise estimation. BLEU scores are shown as percentages.

the generator also needs to learn the alignment of words and phrases to SVPs and determine whether all required information is present on the output (see the introduction to this chapter).

Our differing tree updates clearly bring a substantial improvement over standard perceptron updates, and scores keep increasing with bigger amounts of training data used, whereas with plain perceptron updates, the scores stay flat. The increase with using 100% training data is smaller than previous steps since all training DAs are in fact used twice, each time with a different paraphrase.[10] A larger training set with different DAs should bring a bigger improvement. Using future promise estimation boosts the scores even further, by a smaller amount for BLEU but noticeably for NIST. Both improvements on the full training set are considered statistically significant at 95% confidence level by the paired bootstrap resampling test (Koehn, 2004).

In addition, we performed a manual inspection of two randomly selected cross-validation rounds. This confirmed that the automatic scores reflect the quality of the generated sentences well. If we look closer at the generated sentences (see Table 5.2), it becomes clear that the generator learns to produce

[10] We used the two paraphrases that come with each DA as independent training instances. While having two different gold-standard outputs for a single input is admittedly not ideal for a discriminative learner, it still brings an improvement in our case.

| | |
|---|---|
| Input DA | inform(name=X-name, type=placetoeat, eattype=restaurant, near=X-near, food=Continental, food=French) |
| Reference | X is a French and continental restaurant near X. |
| Generated | X is a French and continental restaurant near X. |
| Input DA | inform(name=X-name, type=placetoeat, area=riverside, near=X-near, eattype=restaurant) |
| Reference | X restaurant is near X on the riverside. |
| Generated | X is a restaurant in the riverside area near X. |
| Input DA | inform(name=X-name, type=placetoeat, area=X-area, pricerange=moderate, eattype=restaurant) |
| Reference | X is a moderately priced restaurant in X. |
| Generated | X is a restaurant in the X area. [moderate] |
| Input DA | inform(name=X-name, type=placetoeat, eattype=restaurant, area=riverside, food=French) |
| Reference | X is a French restaurant on the riverside. |
| Generated | X is a French restaurant in the riverside area which serves French food. |
| Input DA | inform(name=X-name, type=placetoeat, eattype=restaurant, pricerange=moderate, area=X-area, food=Contemporary, food=English) |
| Reference | X is a moderately priced English contemporary restaurant in X. |
| Generated | X is an English restaurant in the X area which serves expensive food in the moderate price range located in X. [Contemporary] |
| Input DA | inform(name=X-name, type=placetoeat, eattype=restaurant, area=citycentre, near=X-near, food="Chinese takeaway", food=Japanese) |
| Reference | X is a Chinese takeaway and Japanese restaurant in the city centre near X. |
| Generated | X is a Japanese restaurant in the centre of town near X and X. [Chinese takeaway] |
| Input DA | inform(name=X-name, type=placetoeat, pricerange=moderate, eattype=restaurant) |
| Reference | X is a restaurant that offers moderate price range. |
| Generated | X is a restaurant in the moderate price range. |

Table 5.2: Example generated sentences

Sentences generated when training on the full training set and using differing subtree updates and future promise estimation. Errors are marked in color (missing, superfluous, repeated information).

meaningful utterances which mostly correspond well to the input DA. It is able to produce original paraphrases and generalizes to previously unseen DAs.

On the other hand, not all required information is always present, and some facts are sometimes repeated or irrelevant information appears (see Table 5.2 for examples). We quantified the semantic errors on our two selected cross-validation rounds, counting missing, superfluous, and repeated information in the outputs. All setups scored similarly: we counted 26 errors in the outputs of the basic perceptron setting, 32 with the addition of differing subtree updates, and 30 with additional future promise estimation.

We could see that the errors mostly happen with input slot-value pairs that occur only rarely in the training data; the small size of the training set is probably at cause here. Another problem is posed by repeated slots in the input DA, which are sometimes not reflected properly in the generated sentence.

## 5.7 Flexibility Issues

In this section, we describe some follow-up experiments where we encountered problems with our A*-search-based generator. These problems eventually led us to adopt a completely different approach based on RNNs.

### Neural Network Scorer

The semantic error issues we encountered in our experiments on the BAGEL dataset could probably be resolved by enriching or further refining the scorer feature set; however, such a solution would require a lot of handcrafting and trial-and-error experiments, and its applicability to other domains might be limited. Therefore, we decided to address the problems by replacing the perceptron scorer with a neural network, thus obviating the need for handcrafted features altogether.

Our approach was straightforward – instead of the perceptron scorer (see Section 5.4), the candidate sentence plan trees were input to a feedforward NN (see Section 1.4) giving out a score:

$$\text{score}(t, d) = \text{nn}_{\mathbf{w}}(t, d) \tag{5.4}$$

In 5.4, $t$ is a candidate sentence plan tree, $d$ is the input DA, and "nn" represents a neural network. The NN parameters were updated very similarly as in the perceptron case: whenever a higher score was assigned to a generation output $t_{top}$ than to the corresponding gold-standard sentence plan tree $t_{gold}$, the NN

parameters $\mathbf{w}$ were updated in a stochastic gradient descent (SGD) fashion (Bishop, 2006, p. 240f.):

$$\mathbf{w} = \mathbf{w} + \alpha \cdot (\frac{\partial}{\partial \mathbf{w}} \text{nn}_{\mathbf{w}}(t_{gold}, d) - \frac{\partial}{\partial \mathbf{w}} \text{nn}_{\mathbf{w}}(t_{top}, d)) \tag{5.5}$$

We experimented with many different network shapes and hyperparameter values, but the results stayed well below that of the perceptron scorer both in terms of automatic scores and in manually assessed quality. We believe that the updates occurring only after a full tree has been generated are probably too infrequent, representing a supervision signal too weak to fully learn all NN parameters.

### Scalability to a Larger Dataset

After Wen et al. (2015b) made their datasets available (see Section 2.4), we tested our generator on their restaurant dataset. We only performed a few preliminary experiments and did not attempt to search for parameters or tweak the feature set extensively, but the first results were rather disappointing. On the much larger dataset, the generator became very slow and the output sentences were not fluent in most cases. The output quality problem could be overcome by adding more features designed for this dataset, and feature extraction could be optimized to some degree, but we decided to focus on the more promising RNN approach (see Chapter 6).

## 5.8 Discussion

In the following, we first compare our generator to previous approaches, then summarize our results from this chapter.

### Comparison to Previous Approaches

Unlike previous approaches to trainable sentence planning (Walker et al., 2001a; Stent et al., 2004; Paiva and Evans, 2005, and other; see Section 2.2), our sentence planner does not require a handcrafted base module and can be fully trained from data.

There have been previous fully trainable NLG systems that approach sentence planning and surface realization in a joint fashion and do not require a handcrafted module (see Section 2.3). However, unlike our solution, they either require fine-grained alignments between MR elements and words/phrases (see

Section 3.2 Mairesse et al., 2010; Dethlefs et al., 2013), or use a special preprocessing step to obtain alignments (Wong and Mooney, 2007; Angeli et al., 2010). In addition, the basic algorithm of these approaches often requires a specific input MR format, e.g., a tree (Wong and Mooney, 2007; Lu et al., 2009) or a flat database (Angeli et al., 2010; Mairesse et al., 2010; Konstas and Lapata, 2013), whereas our generator is not limited in this respect.

Our generator is also the first to use deep syntactic dependency trees in a fully trainable sentence planning setup; previous approaches using deep syntax trees (Bohnet et al., 2010; Belz et al., 2012; Ballesteros et al., 2014) focus solely on surface realization.

Our sentence planning approach is most similar to the work of Zettlemoyer and Collins (2007), which use a candidate generator and a perceptron ranker for CCG parsing. Apart from proceeding in the opposite direction and using dependency trees instead of CCG structures, we only use very generic rules in our candidate generator instead of language-specific ones, and we incorporate differing subtree updates and future promise estimation into our ranker.

A very recent work of Lampouras and Vlachos (2016) directly compares to our approach described here and in (Dušek and Jurčíček, 2015), achieving similar results – slightly worse BLEU and ROUGE (Lin, 2004) scores but slightly better NIST and number of semantic errors. In a human evaluation study, their system was scored lower on naturalness but higher on informativeness than ours.

## Conclusions

Our sentence planner took a novel A*-search-based approach, being the first to use dependency syntax and learn alignment of semantic elements to words or phrases jointly with sentence planning, thus allowing training from unaligned pairs of meaning representations and output utterances.

We have achieved promising results in our experiments on the BAGEL dataset of Mairesse et al. (2010); the utterances produced by our generator were mostly fluent and relevant. They did not surpass the BLEU score of the original authors; however, our task is substantially harder as our generator does not require fine-grained alignments on the input. Our novel features of the sentence planner ranker – using differing subtrees and future promise for perceptron weight updates – have brought significant performance improvements.

However, the outputs of all setups contain a relatively high number of semantic errors (missing, repeated, or irrelevant information) and the features for the scorer, albeit very general, still needed to be designed by hand. Further-

more, our first results on a much larger dataset were rather disappointing, and a straightforward generator extension replacing the perceptron scorer with a feedforward neural network performed poorly.

In our newer experiments (see Chapter 6), the NLG method presented in this chapter has been surpassed by a RNN-based sequence-to-sequence approach in terms of both BLEU/NIST metrics and the number of semantic errors. This newer approach obviates the need for any handcrafted features, and it is more flexible: within the same architecture, it allows to train a sentence planner using deep syntax trees as well as a direct generation of strings. Furthermore, it is able to handle much larger training datasets/domains than BAGEL (see Chapters 7 and 8).

# 6

# Sequence-to-Sequence Generation Experiments

This chapter, based on (Dušek and Jurčíček, 2016b) for the most part, describes our first experiments with the RNN-based sequence-to-sequence (seq2seq) approach to generation. Our new RNN generator is not only able to learn from training data without fine-grained alignments (see Section 3.2), it is also more flexible and faster than the previous approach described in Chapter 5, and yields significantly better results. We are able to compare two-step generation divided into sentence planning and surface realization with a joint, one-step approach (cf. Section 3.4).

In the following, we first introduce and motivate our use of the seq2seq approach in Section 6.1, describing the two different generation modes. Section 6.2 then details the architecture of our generator and the data representations used.

The following two sections deal with our experiments on the BAGEL dataset (Mairesse et al., 2010), where we directly compare our new approach to the generator from Chapter 5. First, we describe our experimental setup in Section 6.3, Section 6.4 then analyzes the results obtained.

The final Section 6.5 then contrasts our generator with other RNN-based approaches to NLG, including a direct experimental comparison to the work of (Wen et al., 2015a), and concludes with a brief summary.

## 6.1 Introduction

With the recent emergence of RNN-based models for various tasks in NLP, most notably seq2seq models with attention for MT (Cho et al., 2014; Sutskever et al., 2014) and first RNN-based NLG approaches (Wen et al., 2015b,a, see Section 2.3), we understood the power of this approach and decided to adapt seq2seq generation to our task. Our new generator uses the seq2seq generation technique combined with beam search and an $n$-best list reranker to suppress irrelevant information in the outputs. The new model is more flexible than most previous solutions including the A*-search-based generator presented in Chapter 5 as it requires neither fine-grained alignments between MR elements and words/phrases in training data (Mairesse et al., 2010), nor a handcrafted base generator (Stent et al., 2004), nor handcrafted features (as our A*-search-based generator).

In addition, our seq2seq generator supports two different generation modes within the same, conceptually simple architecture. As described in Chapters 2 and 3, the task of NLG is traditionally divided into two subtasks: sentence planning (high-level sentence structure decisions), and surface realization (detailed shaping according to target language grammar). NLG systems can be generally divided into two groups (see Section 2.1): Some keep this division and use a two-step pipeline (Walker et al., 2001a; Rieser et al., 2010; Dethlefs et al., 2013, see Section 2.2), others apply a joint, end-to-end model for both tasks (Wong and Mooney, 2007; Konstas and Lapata, 2013, see Section 2.3). Both options offer their advantages: The former simplifies the task by abstracting away from surface grammar, and the latter avoids explicit structure modeling and error accumulation along a pipeline (see Section 3.4). Our new NLG system overrides the division between these two modes by being able to operate in both of them: it either produces natural language strings or generates t-trees (deep syntax dependency trees), which are subsequently processed by the surface realizer described in Chapter 4. This allows us to directly compare two-step generation, where sentence planning and surface realization are separated, with a joint, one-step approach.

Our experiments also show that the seq2seq generator does not necessarily require as large a number of training examples as has been used in most recent RNN-based experiments with NLG (Wen et al., 2015a, 2016c; Mei et al., 2016). We experiment with using much less training data than these approaches, and we find that our system learns successfully to produce both strings and t-trees on the very small BAGEL restaurant information dataset (Mairesse et al., 2010). It is able to surpass $n$-gram-based scores achieved previously by our A*-search-

based generator (see Chapter 5), offering a simpler setup and more relevant outputs.

### Generator Setting

The inputs and outputs to the generator are exactly the same as in the previous chapter (see Sections 3.1 and 5.1), i.e., DAs converted to a list of triplets in the form "DA type – slot – value" and plain text sentences.[1] As mentioned above, our generator operates in two modes, producing either t-trees (see Section 3.5 and Figure 3.5) or natural language strings.

The first mode corresponds to a standalone sentence planner; the generator in this mode is a direct replacement of the sentence planner described in Chapter 5. The same surface realizer as for the previous planner (see Chapter 4 and Section 5.1) is used to linearize the t-trees into natural language strings.

The second generator mode joins sentence planning and surface realization into one step, producing natural language sentences directly.

## 6.2 The Seq2seq Generation Model

Our generator is based on the seq2seq approach (Cho et al., 2014; Sutskever et al., 2014), a type of an encoder-decoder RNN architecture operating on variable-length sequences of tokens (see Section 1.4). In the first step, the encoder RNN consumes the input token by token and encodes it into a hidden state, which is used in the second step by the decoder RNN to produce the output sequence (in a conditioned RNN LM scenario). This approach was originally developed for MT, but it was applied to many other tasks (see Section 2.3). In this section, we first address the necessary conversion of input DA and output trees/sentences into sequences, then describe the main seq2seq component of our generator, which uses a variant of the seq2seq model with attention. Finally, we explain the reranker that supplements the main generator component and uses a similar architecture.

### Sequence Representation of DA, Trees, and Sentences

We represent DA, deep syntax trees, and sentences as sequences of tokens to enable their usage in the sequence-based RNN components of our generator (see

[1] Same as the A*-search-based generator in Chapter 5, adapting the seq2seq-based generator for a MR other than DAs is relatively easy as it only only requires coming up with a sequence input representation of the new MR and a binary coarse-grained representation to be used in the reranker (see Section 6.2).

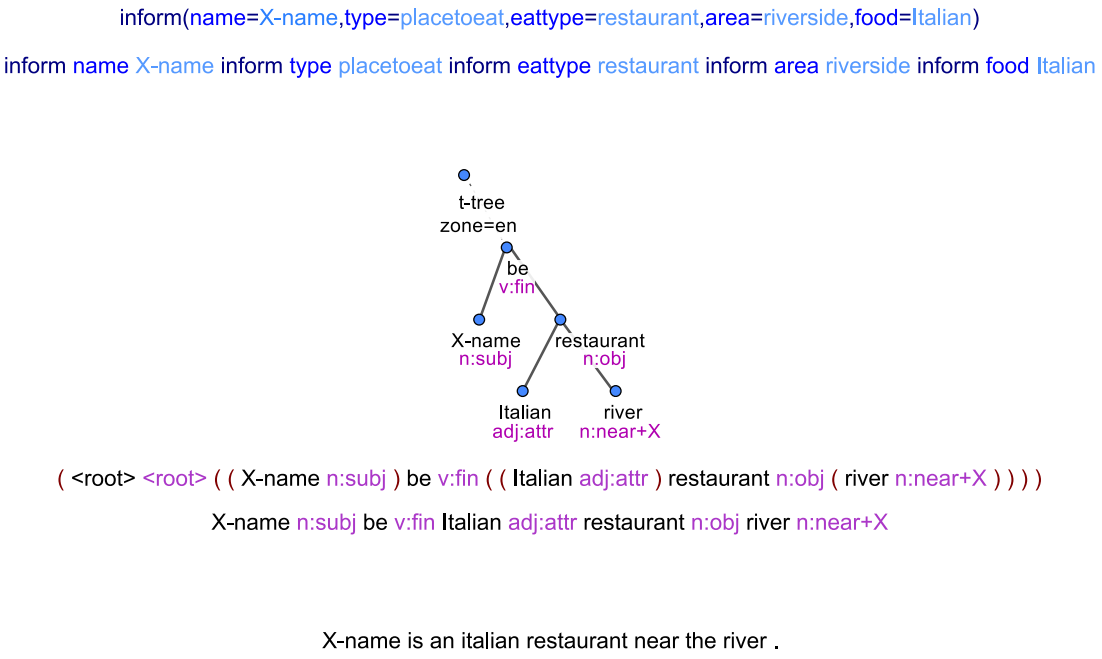


Figure 6.1: Sequence encoding of DAs, sentence plan trees, and output sentences.

Top: DAs, the original DA shown above the encoded sequence. Center: sentence plan trees, showing first the original tree, then its sequence representation in the generator, and finally its representation in the reranker. Bottom: output sentences (tokens are used mostly "as-is").

below). Each token is represented by its embedding – a vector of floating-point numbers (Bengio et al., 2003). Embeddings of the individual tokens are stored in an embedding dictionary/matrix; each distinct token known at training time is assigned an integer ID and the corresponding entry in the matrix. The embeddings are randomly initialized and learned directly from training data.

We use the following sequence representations in our generator:

- *DAs.* To form a sequence representation of a DA, we create a triple of the structure "DA type, slot, value" for each slot in the DA and concatenate the triples (see Figures 6.1 and 6.2).[2]
- *t-trees in the generator.* The t-trees used as sentence plans (the output of the corresponding generator mode) are represented in a bracketed notation similar to the one used by Vinyals et al. (2015a) (see Figure 6.1).

[2]While this may not necessarily be the best way to obtain a vector representation of a DA, it showed to work well in our experiments (see Section 6.4).

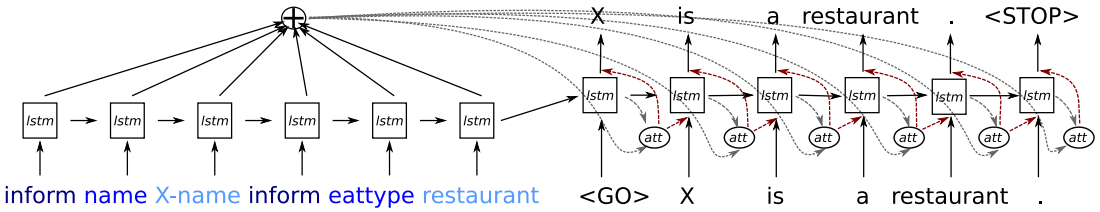


Figure 6.2: The main seq2seq generator with attention.

Left part: encoder, with encoder hidden outputs concatenated to use for the attention model. Right part: decoder; dotted lines indicate data flow in the attention model.

Within its brackets, each node holds its left children, then its lemma (t-lemma)[3] and formeme, and finally its right children.[4]

- *t-trees in the reranker.* In the reranker, structure is disregarded in trees. They are represented as a plain list of lemma-formeme pairs (see Figure 6.1).
- *Surface strings.* Sentences on the output from the seq2seq generator are represented as a simple sequence of words, with the exception of plural nouns, which use the singular noun followed by a special *-s* token. Casing is disregarded (all tokens are lowercased).

All the different representations use separate embedding dictionaries. No dictionary size boundary is enforced, all items encountered in the training data are assigned a dictionary position.

## The Main Seq2seq Generator

Our main generator uses the seq2seq model with attention (Bahdanau et al., 2015; see Figure 6.2).[5] It starts with the encoder stage, which uses an RNN to encode an input sequence $\mathbf{x} = \{x_1, \ldots, x_n\}$ into sequences of hidden outputs $\mathbf{h} = \{h_1, \ldots, h_n\}$ and internal encoder states (cell states) $\mathbf{C} = \{C_1, \ldots, C_n\}$ using the following recurrent function:

$$(h_t, C_t) = \text{lstm}(x_t, h_{t-1}, C_{t-1}) \tag{6.1}$$

[3]Same as in Chapter 5, the deep lemmas or t-lemmas in t-trees are referred to as lemmas here for the sake of brevity as there is no risk of confusion with surface lemmas.

[4]Note that this t-tree representation is simplified as it does not include grammatemes (deep grammatical attributes, see Section 3.5). Most frequent grammateme values found in training data for each lemma-formeme combination are used (cf. Section 5.5).

[5]We use the implementation in the TensorFlow framework (Abadi et al., 2015), version 0.6.

In (6.1), *lstm* stands for the non-linear function represented by a long-short-term memory (LSTM) cell (Hochreiter and Schmidhuber, 1997; Graves, 2013) and consists of the following operations:[6]

$$
\begin{aligned}
f_t &= \sigma(W_f(h_{t-1} \circ x_t) + b_f) && (6.2)\\
i_t &= \sigma(W_i(h_{t-1} \circ x_t) + b_i) && (6.3)\\
o_t &= \sigma(W_o(h_{t-1} \circ x_t) + b_t) && (6.4)\\
C_t &= f_t \cdot C_{t-1} + i_t \cdot \tanh(W_C(h_{t-1} \circ x_t) + b_C) && (6.5)\\
h_t &= o_t \cdot \tanh c_t && (6.6)
\end{aligned}
$$

In the first step, $C_0$ and $h_0$ are zero-initialized. $f_t$ in (6.2) and (6.5) is the so-called *forget gate*, which controls how much of the previous cell state remains into the next step. In (6.3) and 6.5, $i_t$ is the *input gate*, controlling the flow of information from the current input character into the cell state. The *output gate* $o_t$ in (6.4) and (6.6) then controls what information from the cell state is presented in the output (hidden) state $h_t$. $W_f$, $W_i$, $W_o$, and $W_c$ are learned weight matrices, and $b_f$, $b_i$, $b_t$, and $b_C$ are learned bias terms.

The decoder stage then uses the hidden outputs $\mathbf{h}$ and the final encoder cell state $C_n$ to generate a sequence $\mathbf{y} = \{y_1, \ldots, y_m\}$ with a second LSTM-based RNN.[7] The probability of each output token is defined as:[8]

$$p(y_t|y_1, \ldots, y_{t-1}, \mathbf{x}) = \text{softmax}((s_t \circ c_t)W_Y) \tag{6.7}$$

In (6.7), $s_t$ is the decoder hidden output, computed in the following way:

$$(s_t, C'_t) = \text{lstm}((y_{t-1} \circ c_{t-1})W_S, s_{t-1}, C'_{t-1}) \tag{6.8}$$

We can see in (6.8) that the decoder uses the previous output token at each step for predicting the next one. The decoder hidden outputs $s_t$ and cell states $C'_t$ in (6.8) are initialized by the final encoder hidden output and state, i.e., $s_0 = h_n$ and $C'_0 = C_n$. $W_Y$ and $W_S$ are learned linear projection matrices, and "$\circ$" denotes concatenation. $c_t$ represents the *context vector* – a weighted sum of

[6] In equations (6.2) through (6.12), "$\cdot$" stands for element-wise multiplication as opposed to the default matrix/dot product. "$\circ$" then denotes concatenation. $\sigma$ denotes the sigmoid function (1.6), and tanh denotes the hyperbolic tangent function (1.9).
See also `http://colah.github.io/posts/2015-08-Understanding-LSTMs/` (Accessed: November 3, 2016) for a very detailed and graphic explanation of LSTMs.

[7] This is a conditioned RNN LM generation scenario (cf. Section 1.4).

[8] See (1.10) for a definition of the softmax function used in (6.7) and (6.10).

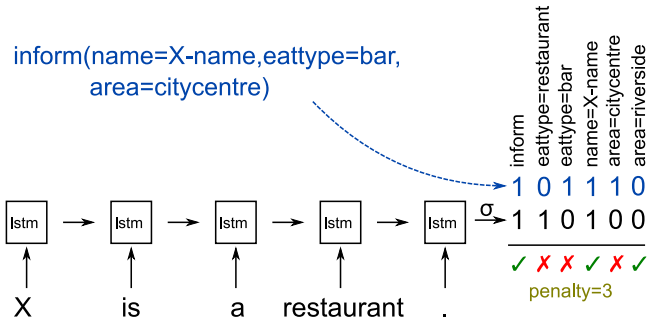


Figure 6.3: The $n$-best list reranker: 1-hot vector classification and comparison with the source DA.

A RNN encoder (bottom left) with a final sigmoid layer classifier ($\sigma$) makes a binary decision on the presence of each of the possible DA items (right). This is compared to the original input DA (shown in blue) and the number of differences constitutes the final penalty score (shown in olive).

the encoder hidden states:

$$c_t = \sum_{i=1}^{n} \alpha_{ti} h_i \tag{6.9}$$

In (6.9), $\alpha_{ti}$ corresponds to an *alignment model*, represented by a feed-forward network with a single tanh hidden layer:[8]

$$\alpha_{ti} = \text{softmax}(v_\alpha^\top \tanh(W_\alpha s_{t-1} + U_\alpha h_i))) \tag{6.10}$$

In (6.10), $v_\alpha$, $W_\alpha$, and $U_\alpha$ are learned weight matrices.[9]

On top of this basic seq2seq model, we implemented a simple beam search for decoding (Sutskever et al., 2014; Bahdanau et al., 2015): In the decoder, the most probable $n$ output sequences at each step are stored and expanded by one further token; the resulting $n$ most probable outputs of this expansion overall are always kept for the next decoder step.

## Reranker

While the main decoder has access to the input DA and is able to produce a relevant output in most cases, it often fails to realize a part of the DA or includes irrelevant information (cf. Section 5.6). To ensure that the output

[9]See also (Luong et al., 2015) for additional explanation of Bahdanau et al. (2015)'s attention model. The tanh function in shown in (1.9).

trees/strings always achieve as close semantic correspondence as possible to the input DA, we implemented a classifier to rerank the $n$-best beam search outputs (see above) and penalize those missing required information and/or adding irrelevant one. Similarly to Wen et al. (2015b)'s convolutional NN reranker, our classifier provides a binary decision for an output tree/string on the presence of all dialogue act types and slot-value combinations seen in the training data, producing a 1-hot vector. The input DA is converted to a similar 1-hot vector (with 1's for present dialogue act types or slot-value combinations, and 0's for those not present). The reranking penalty of the sentence is then the Hamming distance between these two vectors (see Figure 6.3). Weighted penalties for all sentences are subtracted from their $n$-best list log probabilities.

We used binary classification here since a binary penalty is straightforward to interpret and the implementation of such a classifier is very simple. The binary classifier is not able to cover cases where certain slot-value combinations are repeated, but this problem did not occur very often in our preliminary experiments (compared to the A*-search planner described in Chapter 5). Despite this limitation, the reranker still has a very positive effect on the outputs (cf. Section 6.4).

We employ a similar architecture for the classifier as in the encoder of our main seq2seq generator (see above) – we used an LSTM-based RNN encoder operating on the output trees/strings and a single sigmoid layer for classification over the last encoder hidden state. Given an output sequence representing a string or a tree $\mathbf{y} = \{y_1, \ldots, y_n\}$,[10] the encoder again produces a sequence of hidden outputs $\mathbf{h} = \{h_1, \ldots, h_n\}$ and internal cell states $\mathbf{C} = \{C_1, \ldots, C_n\}$ where:

$$(h_t, C_t) = \text{lstm}(y_t, h_{t-1}, C_{t-1}) \tag{6.11}$$

$C_0$ and $h_0$ are zero-initialized. The output binary vector $o$ is computed based on the last hidden output $h_n$ as:

$$o_i = ||\sigma((h_n \cdot W_R + b)_i)|| \tag{6.12}$$

In (6.12), $W_R$ is a learned projection matrix, $b$ is a corresponding bias term, and $\sigma$ represents the sigmoid function (1.6) producing a value in the range $\{0, 1\}$. The $||$ sign represents rounding applied to yield a binary decision.

[10] Depending on the generator mode (cf. Section 6.1). See also this section on page 81 for details on how trees are represented in the reranker encoder.

## 6.3 Experiments

Same as in Chapter 5, we perform our experiments on the freely available BAGEL data set of Mairesse et al. (2010) with 202 DAs and 404 sentences from the restaurant information domain. To avoid data sparsity, we adopt the same partial delexicalization scenario as used by the authors of the dataset and in Section 5.5 (cf. also Section 3.3). Unlike Mairesse et al. (2010) and same as in Chapter 5, we do not use manually annotated alignment of slots and values in the input DA to target words and phrases and let the generator learn it from data, which simplifies training data preparation but makes our task harder (see Section 3.2).

For generating into strings, we just use the plain text sentences (lowercased and with separated plural *-s*, see Section 6.2). As described in Section 3.4, we apply automatic analysis from the Treex NLP toolkit (Popel and Žabokrtský, 2010) to obtain deep syntax trees for training tree-based generator setups.[11] Same as in Chapter 5 and (Mairesse et al., 2010), we apply 10-fold cross-validation with 181 training DAs and 21 testing DAs. In addition, we reserve 10 DAs from the training set for validation. We treat the two paraphrases for the same DA as separate instances in the training set, but we use them together as two references when measuring BLEU and NIST scores (Papineni et al., 2002; Doddington, 2002, see Section 3.6) on the validation and test sets.

To train our seq2seq generator, we minimize unweighted output sequence cross-entropy using the Adam gradient descent optimizer (Kingma and Ba, 2015; Goodfellow et al., 2016, p. 308f.). We set the parameters of our setup based on a few preliminary experiments: The Adam learning rate is set to 0.001, embedding size is 50, LSTM cell size 128, and batch size 20. We perform 10 runs with different random initialization of the network and up to 1,000 passes over the training data, validating after each pass and selecting the parameters that yield the highest BLEU score on the validation set. Training is terminated early (before 1,000 passes) if the top 10 so far achieved validation BLEU scores do not change for 100 passes. Neither beam search nor the reranker are used for validation.

We use the Adam optimizer minimizing cross-entropy to train the reranker as well, and we use the same network and optimization parameters as with the main seq2seq generator. We perform a single run of up to 100 passes over the data. We also validate after each pass, and we select the parameters

[11] Depending on the cross-validation split, the input vocabulary size is around 45 (DA types, slots, and values added up) and output vocabulary sizes are around 170 for string generation and 180 for tree generation (45 formemes and 135 lemmas).

| Setup | BLEU | NIST | SemErr |
|---|---|---|---|
| Mairesse et al. (2010)* | ∼67 | - | 0 |
| Best A*-search-based result (Chapter 5) | 59.89 | 5.231 | 30 |
| Greedy with trees | 55.29 | 5.144 | 20 |
| + Beam search (beam size 100) | 58.59 | 5.293 | 28 |
| + Reranker (beam size 5) | 60.77 | 5.487 | 24 |
| (beam size 10) | 60.93 | 5.510 | 25 |
| (beam size 100) | 60.44 | 5.514 | 19 |
| Greedy into strings | 52.54 | 5.052 | 37 |
| + Beam search (beam size 100) | 55.84 | 5.228 | 32 |
| + Reranker (beam size 5) | 61.18 | 5.507 | 27 |
| (beam size 10) | 62.40 | 5.614 | 21 |
| (beam size 100) | 62.76 | 5.669 | 19 |

Table 6.1: Results of our seq2seq generator on the BAGEL data set

NIST, BLEU, and semantic errors in a sample of the output.

*Mairesse et al. (2010)'s result is not directly comparable to ours due to their usage of manual alignments. The zero semantic error is implied by the manual alignments and the architecture of their system.

giving minimal Hamming distance on both validation and training set, where the validation set is given 10 times more importance. Reranking penalty for decoding is set to 100, in order to penalize semantic errors severely.

## 6.4 Results

The results of our experiments and a comparison to previous works on this dataset are shown in Table 6.1. We include BLEU and NIST scores and the number of semantic errors measured in the same way as in Section 5.6.[12]

**Joint, end-to-end setup.** The outputs of direct string generation show that the models learn to produce fluent sentences in the domain style;[13] incoherent sentences e.g., *a restaurant serving Japanese restaurant near X*, are rare, but semantic errors are very frequent in the greedy search. Most errors involve confusion of semantically close items, e.g., *Italian* instead of *French* or *riverside area* instead of *city centre* (see Table 6.2); items occurring more frequently are preferred

[12]We manually counted all incorrect, missing, and repeated pieces of information on the outputs of two randomly selected cross-validation runs.

[13]The average sentence length is around 13 tokens.

| | |
|---|---|
| Input DA | inform(name=X-name, type=placetoeat, eattype=restaurant, area=citycentre, near=X-near, food="Chinese takeaway", food=Japanese) |
| Reference | X is a Chinese takeaway and Japanese restaurant in the city centre near X. |
| Greedy with trees | X is a restaurant offering chinese takeaway in the centre of town near X. [Japanese] |
| + Beam search | X is a restaurant and japanese food and chinese takeaway. |
| + Reranker | X is a restaurant serving japanese food in the centre of the city that offers chinese takeaway. |
| Greedy into strings | X is a restaurant offering italian and indian takeaway in the city centre area near X. [Japanese, Chinese] |
| + Beam search | X is a restaurant that serves fusion chinese takeaway in the riverside area near X. [Japanese, citycentre] |
| + Reranker | X is a japanese restaurant in the city centre near X providing chinese food. [takeaway] |
| Input DA | inform(name=X-name, type=placetoeat, eattype=restaurant, area=riverside, food=French) |
| Reference | X is a French restaurant on the riverside. |
| Greedy with trees | X is a restaurant providing french and continental and by the river. |
| + Beam search | X is a restaurant that serves french takeaway. [riverside] |
| + Reranker | X is a french restaurant in the riverside area. |
| Greedy into strings | X is a restaurant in the riverside that serves italian food. [French] |
| + Beam search | X is a restaurant in the riverside that serves italian food. [French] |
| + Reranker | X is a restaurant in the riverside area that serves french food. |
| Input DA | inform(name=X-name, type=placetoeat, eattype=restaurant, near=X-near, food=Continental, food=French) |
| Reference | X is a French and Continental restaurant near X. |
| Greedy with trees | X is a french restaurant that serves french food and near X. [Continental] |
| + Beam search | X is a french restaurant that serves french food and near X. [Continental] |
| + Reranker | X is a restaurant serving french and continental food near X. |
| Greedy into strings | X is a french and continental style restaurant near X. |
| + Beam search | X is a french and continental style restaurant near X. |
| + Reranker | X is a restaurant providing french and continental food, near X. |

Table 6.2: Example outputs of different seq2seq generator setups.

Beam size is set to 100 in all beam search and beam search + reranker setups used to produce the outputs shown. Errors are marked in color (missing, superfluous, repeated information, disfluency).

regardless of their relevance. The beam search brings a BLEU improvement but keeps most semantic errors in place. The reranker is able to reduce the number of semantic errors while increasing automatic scores considerably, surpassing the best results on BAGEL data without alignments achieved by us in Chapter 5. Using a larger beam increases the effect of the reranker as expected, resulting in slightly improved outputs.

**2-step setup with t-trees.** Models generating deep syntax trees are also able to learn the domain style, and they have virtually no problems producing valid trees.[14] The surface realizer works almost flawlessly on this limited domain (see Section 5.1), which guarantees outputs that are fluent on the surface and leaves the seq2seq generator as the major error source. The syntax-generating models tend to make different kinds of errors than the string-based models: Some outputs are valid trees but not entirely syntactically fluent; missing, incorrect, or repeated information is more frequent than a confusion of semantically similar items (see Table 6.2). The models sometimes seem to prefer correct syntactic structure to semantic relevance. Semantic error rates of greedy and beam-search decoding are lower than for string-based models, partly because confusion of two similar items counts as two errors (for one missing and one irrelevant item). The beam search brings an increase in BLEU but also in the number of semantic errors. The reranker is able to reduce the number of errors and improve automatic scores slightly. A larger beam leads to a small BLEU decrease even though the sentences contain less errors; here, NIST reflects the situation more accurately.

**Comparison.** A comparison of the two approaches goes in favor of the joint setup: Without the reranker, models generating trees produce less semantic errors and gain higher BLEU/NIST scores. However, with the reranker, the string-based model is able to reduce the number of semantic errors while producing outputs significantly better in terms of BLEU/NIST.[15] In addition, the joint setup does not need an external surface realizer. The best results of both setups surpass the previous best results on this dataset using training data

[14] The generated sequences are longer, but have a very rigid structure, i.e., less uncertainty per generation step. The average output length is around 36 tokens in the generated sequence or 9 tree nodes; surface realizer outputs have a similar length as the sentences produced in direct string generation.

[15] The difference is statistically significant at 99% level according to pairwise bootstrap resampling test (Koehn, 2004).

without manual alignments, achieved in Chapter 5, in both automatic metrics[16] and the number of semantic errors.

## 6.5 Discussion

In the final section of this chapter, we first compare our new seq2seq-based generator to other recent RNN-based generators, then offer a few summarizing and concluding remarks.

### Comparison to Other RNN-based NLG Systems

The new seq2seq-based generator retains the advantages of the A*-search-based generator over previous approaches, as described in Section 5.8: it is fully trainable and does not require a base handcrafted module, and it can be trained to generate from data without fine-grained alignments or an alignment preprocessing step. In addition, it does not require handcrafted features and it is capable of working both as a standalone sentence planner and as a joint, one-step generator.

Other recent RNN-based generators are also capable of generating from unaligned data. Wen et al. (2015b)'s RNNs with a convolutional network reranker and Wen et al. (2015a)'s gated LSTM generators are very similar to our work but use a hardwired 1-hot encoding of the inputs to the network, which limits the system's flexibility regarding the input MR format. Furthermore, both generators use heuristic error checking based on slot placeholders in the output and thus require fully delexicalized datasets for generation.[17] Mei et al. (2016) present the only seq2seq-based NLG system known to us, but their focus is mostly on content selection. As content selection is handled by the dialogue manager in a SDS (cf. Section 2.1), we do not include it in our model. We extend the previous RNN-based generation experiments by generating deep syntax trees as well as strings and directly comparing pipeline and joint generation within a single architecture. In addition, we experiment with an order-of-magnitude smaller dataset than other RNN-based systems, and we show that our system can be trained to produce meaningful outputs with very little data.

We trained our system on the restaurant dataset of Wen et al. (2015a) in order to perform a direct comparison of our system's performance to theirs. We

[16] The BLEU/NIST differences are statistically significant according to the pairwise bootstrap resampling test.

[17] Our approach allows delexicalization but does not require it. We use partial delexicalization for the BAGEL set (see Section 3.3).

| System | BLEU | ERR |
|---|---|---|
| Wen et al. (2015a) | 73.1 | 0.46 |
| Ours | 72.7 | 0.41 |

Table 6.3: A direct comparison of our system against the results of Wen et al. (2015a) on their restaurant information dataset.

All numbers are percentages. The results are averaged over five different random network initializations.

did not change the architecture of our the system,[18] apart from supporting a random selection from the top 5 $n$-best list outputs in order to make our system's behavior similar to Wen et al.'s and allow a direct comparison. Same as Wen et al., we measured BLEU using a random selection from the 5 top outputs and average slot error rate ERR over all of the 5 top outputs (see Section 3.6). We used five different random initializations of our networks and averaged the results.[19] The results are shown in Table 6.3; we can see that our system performs comparably to Wen et al.'s, offering slightly lower BLEU scores but slightly lower slot error rate.

## Conclusions

We have presented a direct comparison of two-step generation via deep syntax trees with a direct generation into strings, both using the same NLG system based on the seq2seq approach. While both approaches offer decent performance, their outputs are quite different. The results show the direct approach as more favorable, with significantly higher $n$-gram based scores and a similar number of semantic errors in the output.

We also demonstrated that our generator can learn to produce meaningful utterances using a much smaller amount of training data than what is typically used for RNN-based approaches. The resulting models had virtually no problems with producing fluent, coherent sentences or with generating valid structure of bracketed deep syntax trees. Our generator was able to surpass the best BLEU/NIST scores on the same dataset previously achieved by

[18] We did not even change any parameters of our network but for the Adam learning rate and the beam size, which were lowered to 0.0005 and 20, respectively, based on preliminary experiments on development data.

[19] This is different than our previous experiments on the BAGEL set, where we select just one network based on development data performance. Here we proceed in the same way as Wen et al. (2015a).

our perceptron-based generator (see Chapter 5) while reducing the amount of irrelevant information on the output. In addition, it performed comparably to state-of-the-art on a larger dataset. We continue to add improvements to our seq2seq generator in our following experiments; contextual awareness is described in Chapter 7 and enhancements to lexicalization and morphological capabilities are dealt with in Chapter 8.

# 7

# Generating User-adaptive Outputs

In this chapter, which is based on (Dušek and Jurčíček, 2016a) and (Dušek and Jurčíček, 2016c), we concern ourselves with the phenomenon of entrainment (alignment) in dialogue: We enable our seq2seq system from Chapter 6 to adapt to the user's way of speaking, thus providing contextually appropriate, more natural, and possibly more successful output.

As there is no previous dataset for NLG in SDSs available that would take user utterances into account (cf. Section 2.4), we collected a new training dataset using crowdsourcing. We then successfully modified the seq2seq generator architecture to support adapting to previous user utterance, thus producing contextually aware and user-adaptive outputs. We show that our context-aware solution is able to achieve mild but statistically significant gains both in terms of automatic metrics and in human pairwise preference scores.

The text of this chapter is structured in the following way: First, we explain in more detail the notion of entrainment in dialogue in Section 7.1, thus motivating our experiments. Section 7.2 then gives a broad overview over our approach to entrainment-aware NLG and our goals in the following experiments.

The following two sections are dedicated to our novel, context-aware dataset for NLG in the public transport information domain. In Section 7.3, we describe the data collection process; Section 7.4 then summarizes the dataset properties.

Sections 7.5 and 7.6 detail our new context-aware extensions to the seq2seq generator presented in Chapter 6 and our experiments on our newly collected dataset, respectively.

In the final Section 7.7, we briefly compare our context-aware generator to previous approaches to entrainment-capable NLG, and close with a short summary and a few improvement ideas.

## 7.1 Entrainment in Dialogue

In a conversation, speakers are influenced by previous utterances of their counterparts and tend to adapt their way of speaking to each other, reusing lexical items as well as syntactic structure and prosody (Reitter et al., 2006; Levitan, 2014; see Table 7.4). This phenomenon of mutual linguistic convergence of speakers over the course of a conversation is referred to as *entrainment* or *dialogue alignment*. Entrainment occurs naturally and subconsciously and facilitates successful conversations (Friedberg et al., 2012). In addition, entrainment forms a natural source of variation in dialogues.

Nenkova et al. (2008) have shown that higher entrainment in frequent words correlates with a higher success rate in human-human task-oriented dialogues. Users have been reported to entrain naturally to natural language computer interfaces, such as the prompts of a SDS (e.g., Karlgren, 1992; Stoyanchev and Stent, 2009; Parent and Eskenazi, 2010).

The NLG component of a SDS has a great impact on the perceived naturalness of the whole system; NLG quality can also influence the overall task success (Stoyanchev and Stent, 2009; Lopes et al., 2013). However, typical NLG systems in SDSs only take the input DA into account and have no way of adapting to the user's way of speaking. To avoid repetition and add certain amount of variation into the outputs, handcrafted systems typically alternate between a handful of preset variants (Jurčíček et al., 2014) and trainable systems employ overgeneration and random sampling from a $k$-best list of outputs (Wen et al., 2015a). This is perceived as more natural by the users but still does not reach the level of naturalness of human-human dialogues due to lack of adaptation to previous context.

There have been several attempts to introduce two-way entrainment into SDS, i.e., to let the system entrain to user utterances. Hu et al. (2014) report an increased naturalness of the system responses, while Lopes et al. (2013) and Lopes et al. (2015) also mention increased task success. All of these approaches so far focus only on lexical entrainment and are completely or partially rule-based.

## 7.2 Our Approach to Entrainment-Capable NLG

We attempt to take entrainment even further in the context of a fully trainable NLG and train a system that adapts to users' lexical as well as syntactic choices.

inform( line=M102, direction="Herald Square", vehicle=bus,
departure_time=9:01am, from_stop="Wall Street")
*Take bus line M102 from Wall Street to Herald Square at 9:01am.*

is there another option
inform( line=M102, direction="Herald Square", vehicle=bus,
departure_time=9:01am, from_stop="Wall Street")
*There is a bus at 9:01am from Wall Street to Herald Square using line M102.*

Figure 7.1: A comparison of an ordinary NLG training instance (top) and a context-aware one (bottom).

The top instance only includes the input DA and the output sentence; no context is taken into account. The bottom instance includes the preceding user utterance (context), the input DA, and a context-appropriate output sentence. Entrainment (reuse of syntactic and lexical items) in the bottom sentence is marked in color.

We hope that this will lead to further increases in both perceived naturalness of the system responses and the overall task success rate.

In order to build and a test a new context-aware fully trainable NLG system, we collected an NLG dataset for SDSs that is, to our knowledge, the first dataset of its kind to include preceding context (user utterance) with each data instance (source DA and target natural language paraphrase to be generated, see Figure 7.1). Crowdsourcing was used to obtain both the contextual user utterances and the corresponding system responses to be generated. The dataset covers the domain of public transport information, but the method used to collect the data is completely domain-independent; we believe that the results obtained on this dataset are applicable to other domains as well as open-domain and chat-oriented systems. The dataset is is released under a permissive Creative Commons 4.0 BY-SA license for further proof-of-concept experiments studying entrainment in human-computer dialogues.

Building upon the seq2seq generator described in Chapter 6 and using the collected dataset, we created a novel, fully trainable context-aware NLG system for SDSs that is able to entrain to the user and provides naturally variable outputs because its generation process is conditioned not only on the input DA, but also on the preceding user utterance. It is, to our knowledge, the first fully trainable entrainment-enabled NLG system for SDSs. The results of our experiments on the collected dataset show that our context-aware system

outperforms the baseline in both automatic metrics and a human pairwise preference test.

## 7.3 Collecting a Context-aware NLG Dataset

When collecting our dataset, we aimed at capturing entrainment between pairs of user utterances and system responses as close as possible to the one naturally occurring in human-human dialogues. Collecting complete natural human-human task-oriented dialogues would probably yield better conditions for entrainment and make much wider contexts available in our dataset. However, in order to avoid data sparsity, we limited our view of context to a single preceding user utterance, which is likely to have the largest entrainment influence.

To obtain both natural user utterances and natural system responses, we took the following approach:

1. User utterances were recorded in calls to a live SDS running on a toll-free telephone number.
2. The recorded utterances were manually transcribed.
3. The transcriptions were parsed and delexicalized.
4. Based on the meaning of the recorded user utterances, we generated possible response DAs.
5. We obtained natural language paraphrases for the generated response DAs in the context of the collected user utterances.

In order to collect diverse and natural user utterances in a fast and affordable way, we recruited untrained speakers on the the CrowdFlower (CF) crowdsourcing platform[1] to perform call recording, transcription, and response paraphrase creation. To attract native speakers only, the tasks were only made available to CF workers in English-speaking countries.[2]

In the following, we first further describe the domain we used for our dataset and the SDS employed for recording calls. We then give details on the individual data collection steps listed above.

[1] `http://crowdflower.com` (Accessed: March 19, 2017).

[2] We only chose states with a native English-speaking majority: United States, Canada, United Kingdom, Ireland, Australia, New Zealand. Since the toll-free number was U.S.-based, the vast majority of callers came from the U.S.

## The Domain

To collect our dataset, we used the domain of English public transport information as implemented in the Alex SDS framework (Jurčíček et al., 2014; Vejman, 2015) because it was readily available to us. Alex is a mixed-initiative dialogue system, and for this domain, it uses the Google Maps API[3] to find public transit directions among bus and subway stops in New York City. The user is able to specify a time preference or select a means of transport; they may also ask for the duration or distance of the trip.

The Alex system is also capable of providing weather and time information; however, to prevent data sparsity, we limited the domain for our experiments just to public transport among bus and subway stops on Manhattan.

## Recording Calls

Using the Alex English public transport information SDS, we recorded calls in a setting similar to SDS user evaluation (Jurčíček et al., 2011).[4] CF workers were given a toll toll-free phone number to call and a prose description of tasks that they should attempt to achieve. If they completed at least five turns, the SDS would give them a code that allows them to collect CF reward.[5]

The task descriptions presented to the users were designed so that variable and natural utterances are obtained. Even though the task itself stayed relatively similar,[6] we varied the description and used different synonyms (e.g., *schedule/ride/connection*) so that the users are primed with varying expressions. In addition, the description was deliberately written as free text paragraph rather than a list of subtasks (see Figure 7.2). To generate the task descriptions, we used Alex's template NLG system with a specially-designed set of templates where many combinations can be created at random.

Furthermore, the task assignment was presented to the users as a SDS evaluation – they were not aware that the exact wording of their requests is important, and they were not penalized in any way for deviating from the task assigned to them. However, according to manual cursory checks of the recordings, the users mostly tried to complete the task assigned to them and often kept to the wording given to them in the description.

[3] `https://developers.google.com/maps/` (Accessed: March 19, 2017).

[4] The CF task design was adapted from Vejman (2015)'s evaluation tasks.

[5] We used AJAX calls to our dedicated server to check code validity from within the CF task.

[6] The users were supposed to ask for directions between two stops and request several additional details, such as the duration of the ride, or ask for a schedule at a different time.

You want a connection – your departure stop is *Marble Hill*, and you want to go to *Roosevelt Island*. Ask how long the journey will take. Ask about a schedule afterwards. Then modify your query: Ask for a ride at six o'clock in the evening. Ask for a connection by bus. Do as if you changed your mind: Say that your destination stop is *City Hall*.

---

You are searching for transit options leaving from *Houston Street* with the destination of *Marble Hill*. When you are offered a schedule, ask about the time of arrival at your destination. Then ask for a connection after that. Modify your query: Request information about an alternative at six p.m. and state that you prefer to go by bus.

---

Tell the system that you want to travel from *Park Place* to *Inwood*. When you are offered a trip, ask about the time needed. Then ask for another alternative. Change your search: Ask about a ride at 6 o'clock p.m. and tell the system that you would rather use the bus.

Figure 7.2: Examples of task assignments for the calls to the Alex English public transport information SDS.

Notice that the assignments are very similar but use different synonyms (*connection*, *transit options*, *travel* etc.). Also notice the prose form that should allow freer interpretation than a numbered list of subtasks.

We collected 177 calls comprising 1,636 user utterances. We decided to also include into our dataset the recordings collected previously by Vejman (2015) (347 calls and 2,530 utterances). The response generation step (see below) then selected 630 utterances from our calls and 384 utterances from the calls of Vejman (2015) that were relevant to our limited domain.[7]

## Transcription

In order to include accurate contexts in our dataset and to ensure that the corresponding system responses are relevant, we had our recorded calls manually transcribed using the standard CF transcription task. A brief description of the domain and lists of frequent words/expressions and subway stations were provided to transcribers to minimize the number of errors. To ensure transcription

[7] As Vejman (2015)'s calls involve a broader domain and a less varied but more vague task description had been used to collect them, many utterances are irrelevant for our dataset, but a part is still useful.

quality, we used CF's test questions (a set of pre-transcribed sentences on which the CF workers are tested). In addition, simple automatic checks for illegal characters in the transcriptions (e.g., complex punctuation) were performed online via JavaScript.

We collected three transcriptions per utterance and used the transcription variant provided by at least two users, resolving a small number of problematic cases manually.

## Re-parsing

We needed to identify the meaning of the transcribed user utterances in order to generate relevant system response DAs (see below). While the recorded calls contain Spoken Language Understanding (SLU) parses of all user utterances, those are based on speech recognition transcriptions. We applied the rule-based Alex SLU system again to manual transcriptions in order to obtain more reliable parses. In addition, we adjusted the rules in the SLU system to fix several errors found in manual cursory checks of its output.

To reduce data sparsity, we delexicalized the utterances based on their SLU parses – all stop names as well as time expressions and names of transport vehicles were replaced with placeholders (see Section 3.3). Identical delexicalized utterances are treated as a single utterance (one context instance) in the dataset, but the information about their frequency is retained.

## Generating response DA

We have created a trivial rule-based bigram policy to generate all possible system response DAs given a preceding DA corresponding to the SLU parse of a context user utterance.[8] Based on the given preceding DA, it is able to generate several types of response DAs:

**iconfirm:** a confirmation that the system understands the utterance

**inform:** an answer, providing a transport connection or specific details

**inform_no_match:** an apology stating that the specified connection cannot be found

**request:** a request for additional information to complete search

Response DAs of the type iconfirm may further be combined with inform or request DAs into a single response DA. As our policy is only able to react to our

[8] In a real dialogue, the correct response would depend on the whole dialogue history.

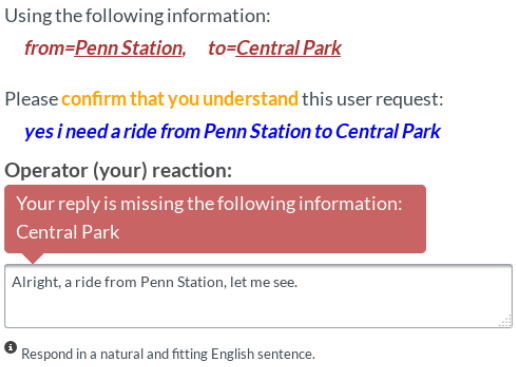


Figure 7.3: A response task in the CrowdFlower interface

Note that the "slot labels" provided to the users are minimalist (here they consist only of prepositions), and when reading the assignment top-to-bottom, that the user first reads the values, then the DA type (intent, sentence type), and finally the context utterance, immediately before starting to create the reply. The figure shows an invalid reply entered and an error message indicating a missing value.

limited domain (see above), it implicitly filters out all irrelevant user utterances: We discard user utterances for which no possible response is produced.

## Obtaining response paraphrases for NLG

The generated response DAs were then used as the input to a CF task where users were asked to create appropriate natural language paraphrases. The CF task was designed with the aim that the collected paraphrases reflect the context of the particular user utterance and are relevant. We performed several trial data collection runs, examined the results, and adjusted the interface accordingly.

The CF worker is asked to write a response of a certain kind (corresponding to DA types listed above) and given information (slots and values) to back it up. The context user utterance is displayed directly above the text entry area to maximize entrainment influence (see Figure 7.3). This simulates a natural situation where a hotline operator hears a request and responds to it immediately. To avoid priming CF workers with slot names (e.g., from_stop, departure_time), we left out slot names where the meaning is unambiguous from the value (e.g., in time expressions) and used very short descriptions (e.g., from, to) elsewhere.[9] The main CF task instructions are relatively short and do

[9]We also experimented with using pictographs instead of textual descriptions, but they proved to be rather confusing to CF workers on our domain.

not include any response examples so that CF workers are not influenced by them.[10] Note that CF workers were not told explicitly to entrain to the previous utterance as we aimed at obtaining a level of entrainment close to the one encountered in natural dialogues.

We use a JavaScript checker directly within the CF task to ensure that the paraphrase contains all required information (the exact value for stop names or time, or one of several synonyms in other slots) and to guide the user towards this goal. We also check for presence of irrelevant information, such as stop names, time expressions, or transport vehicles not included in the assignment.[11] To check the created responses for fluency, we use AJAX calls to our spell-checking server based on Hunspell.[12]

Since about 20% of the responses collected in the testing runs contained errors (irrelevant information or non-fluent responses not discovered by our checks), we performed a manual quality control of all collected responses and requested additional paraphrases on CF where needed. The overhead in the whole dataset stayed around 20%. The manual checking is quite straightforward and manageable given the size of our dataset; for larger datasets, crowdsourcing could also be used for quality control (Mitchell et al., 2014).

## 7.4 Dataset Properties

The dataset has been created over the course of three months, with an estimated net data collection time of one month. The CF task prices as well as total cost of the individual steps are shown in Table 7.3. The final data size statistics are shown in Table 7.1. There are 1,859 pairs of (delexicalized) context user utterances and system response DAs in total, with three natural language paraphrases per pair. The set contains 83 different (delexicalized) system response DAs, which is lower than similar NLG datasets (see Section 2.4), but sufficient for covering our limited domain; the 552 distinct context utterances provide ample space for entrainment experiments. Statistics of the different DA types used in the dataset are given in Table 7.2. By far the most frequent kind of

[10] A testing run with response examples did not bring a better quality of the responses.

[11] In our testing runs, CF workers would often fabulate irrelevant information and include it in their responses.

[12] `http://hunspell.github.io` (Accessed: November 20, 2016). We allowed one typo/unknown word per 10 words at most.

Alternatively, fluency could be also checked using language model scoring, but we found the spellchecker to be a sufficient means of discouraging most users from writing non-fluent sentences or complementing the slot values with random chunks of characters.

| Quantity | Value |
|---|---|
| total response paraphrases | 5,577 |
| unique (delexicalized) context + response DA | 1,859 |
| unique (delexicalized) context | 552 |
| unique (delexicalized) context with min. 2 occurrences | 119 |
| unique response DAs | 83 |
| unique response DA types | 6 |
| unique slots | 13 |

Table 7.1: Dataset size statistics.

| DA type | count |
|---|---|
| inform_no_match | 380 |
| iconfirm | 403 |
| iconfirm&inform | 23 |
| iconfirm&request | 252 |
| inform | 549 |
| request | 252 |

Table 7.2: System response DA type counts in the dataset.

system response in the set is inform with transport directions, which occurs with 380 different context utterances.

Using a portion of the collected data, we assessed the subjective level of entrainment based on word and phrase reuse, word order, or pronominal usages. We estimated that around 59% response paraphrases are syntactically aligned to context utterances, around 31% reuse their lexical items, and around 19% show both behaviors (see Table 7.4 for examples of the different kinds of entrainment).

The dataset is released in CSV and JSON formats and includes the following for each of the 1,859 instances (in both lexicalized and delexicalized form):

- context user utterance,
- occurrence count of the user utterance in recorded calls,
- SLU parse of the user utterance,
- generated system response DA,
- three natural language paraphrases of the system response.

| Task type | Job cost | Total cost |
|---|---|---|
| Calls to the live SDS | $0.20-0.30 | $45.36 |
| Call transcription | $0.05 | $72.24 |
| Providing natural language paraphrases | $0.10-0.20 | $236.85 |

Table 7.3: Crowdsourcing costs for the collection of our dataset.

One call job represents one phone call to the Alex SDS with at least five turns completed. One transcription job consists of five utterances. One paraphrase job comprised creating paraphrases for five different DAs. In total, we collected three transcriptions per sentence and three paraphrases per DA (or more, if some of them were discarded in manual checks).
The costs of calls and transcriptions do not include the calls and transcriptions collected by (Vejman, 2015); both tasks would be approximately twice as expensive if we collected all calls ourselves.

| | |
|---|---|
| Context utterance: | how bout the next ride |
| Response DA: | inform_no_match(alternative=next) |
| Response paraphrases: | Sorry, I did not find a later option.<br>I'm sorry, the next ride was not found. |
| Context utterance: | what is the distance of this trip |
| Response DA: | inform(distance=10.4 miles) |
| Response paraphrases: | The distance is 10.4 miles.<br>It is around 10.4 miles.<br>The trip covers a distance of 10.4 miles. |

Figure 7.4: Entrainment examples from our dataset.

Entraining elements marked in color: lexical entrainment, syntactic entrainment, both lexical and syntactic entrainment.

The dataset has been released under the Creative Commons 4.0 BY-SA license at the following URL:

`http://hdl.handle.net/11234/1-1675`

An updated version with minor corrections is available on GitHub under the following URL:

`https://github.com/UFAL-DSG/alex_context_nlg_dataset`

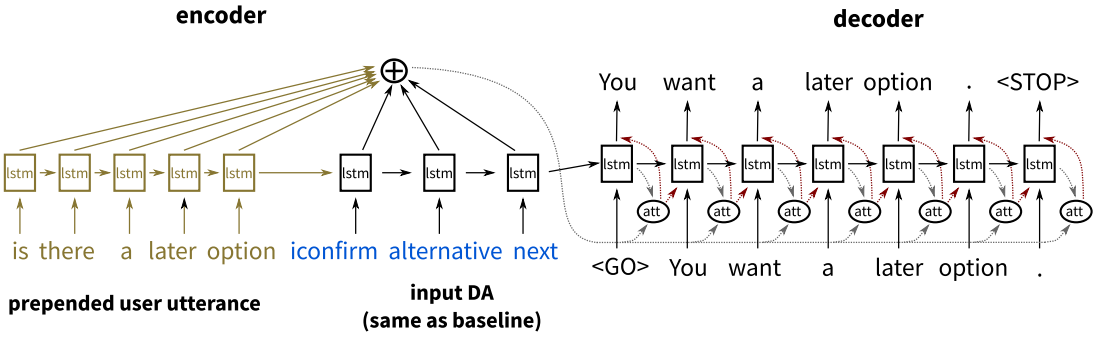


Figure 7.5: The context prepending variant of our generator (with the base seq2seq model shown in black and the new addition highlighted in gold).

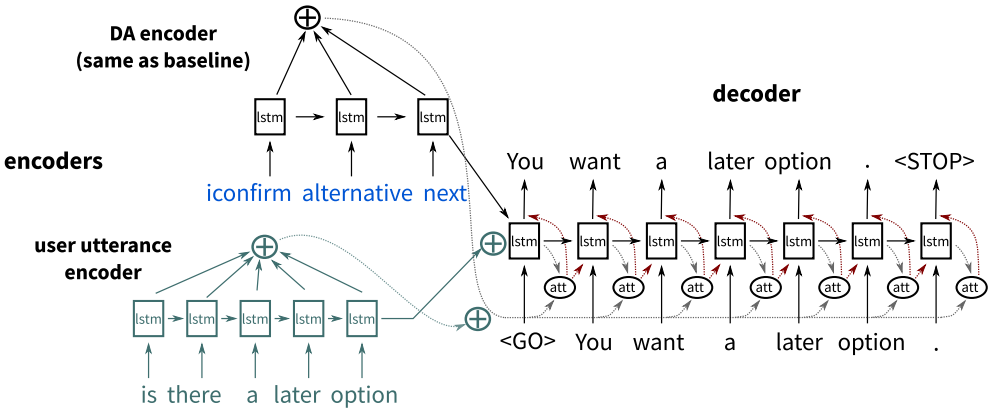


Figure 7.6: The variant of our generator with separate context encoder (the base seq2seq model is shown in black, the new addition is highlighted in teal).

## 7.5 Our Context-aware Seq2seq Generator

Our context-aware seq2seq generator is an improved version of the seq2seq generator described in Chapter 6; therefore, we mostly concentrate here on the context-aware modifications and summarize the seq2seq baseline only briefly.

### Baseline Seq2seq NLG with Attention

As described in Section 6.2, the baseline seq2seq generator has two stages, both composed of LSTM RNN cells. The first, encoder stage encodes the embedding (fixed-size vector) representation of input tokens into a sequence of hidden states. DAs are represented as sequences of triples "DA type, slot, value" on the

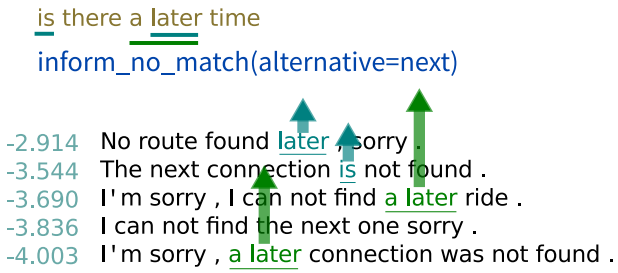


Figure 7.7: The $n$-gram match reranker.

The preceding user utterance (olive) and the input DA (blue) are shown on top. The $k$-best list of system outputs with log probabilities is shown in the bottom, with unigram and bigram matches against the preceding user utterance highlighted in teal and green, respectively.

encoder input, same as in Chapter 6. The second, decoder stage then uses these hidden states to generate the output sequence of tokens. The decoder employs a LSTM RNN initialized by the last encoder hidden state and accompanied by an attention model, which provides access to all encoder hidden states, weighed by their importance for the current generation step.

Note that based on results from Chapter 6, where directly generating strings performed better than a two-step setup generating t-trees (see Section 3.5) and using a separate surface realizer (see Chapter 4), we decided to only use direct string generation in the experiments of this chapter. The tokens generated by the seq2seq decoder are thus directly the words and punctuation marks of a natural language sentence.

Same as in Chapter 6, we use beam search decoding which keeps track of top $k$ most probable output sequences at each generation time step, and we also apply the content classification reranker for the $k$-best lists to penalize irrelevant or missing information on the output (cf. Section 6.2).

## Making the Generator Context-aware

We implemented three different modifications to our baseline generator that make its output dependent on the preceding context:[13]

[13]For simplicity, we kept close to the basic seq2seq architecture of the generator; however, other possibilities for encoding the context, such as using convolution and/or max-pooling (Goodfellow et al., 2016, p. 330ff.), are possible.

**Prepending context.** The tokens of the preceding user utterance are simply prepended to the DA tokens and fed into the encoder (see Figure 7.5). The dictionary for context utterances is distinct from the DA tokens dictionary.

**Context encoder.** We add another, separate encoder for the context utterances. The hidden states of both encoders are concatenated, and the decoder then works with double-sized vectors both on the input and in the attention model (see Figure 7.6).

**$n$-gram match reranker.** We added a second reranker for the $k$-best outputs of the generator that promotes outputs which have a word or phrase overlap with the context utterance (see Figure 7.7). We use geometric mean of modified $n$-gram precisions (with $n \in \{1, 2\}$) as a measure of context overlap, i.e., BLEU-2 (Papineni et al., 2002, see Section 3.6) without brevity penalty. The log probability $l$ of each output sequence on the generator $k$-best list is updated as follows:

$$l = l + w \cdot \sqrt{p_1 p_2} \tag{7.1}$$

In (7.1), $p_1$ and $p_2$ are unigram and bigram precisions of the output sequence against the context (see Eq. 3.2 in Section 3.6), and $w$ is a preset weight. We believe that any reasonable measure of contextual match would be viable here, and we opted for modified $n$-gram precisions because of simple computation, well-defined range, and the relation to the de facto standard BLEU metric.[14] We use BLEU-2 instead of the default BLEU-4, i.e., only unigrams and bigrams, to promote especially the reuse of single words or short phrases.

In addition, we combine the $n$-gram match reranker with both of the two former approaches.

We used gold-standard transcriptions of the immediately preceding user utterance in our experiments in order to test the context-aware capabilities of our system in a standalone setting; in a live SDS, 1-best speech recognition hypotheses and longer user utterance history can be used with no modifications to the architecture.

## 7.6 Experiments

We perform our experiments on the public transport information dataset we have collected (see Sections 7.3 and 7.4). For our experiments, we split the set

[14] We do not use brevity penalty as we do not want to demote shorter output sequences. However, adding it to the formula in our preliminary experiments yielded similar results to the ones presented here.

| Setup | BLEU | NIST |
|---|---|---|
| Baseline (context not used) | 66.41 | 7.037 |
| $n$-gram match reranker | 68.68 | 7.577 |
| Prepending context | 63.87 | 6.456 |
| + $n$-gram match reranker | 69.26 | 7.772 |
| Context encoder | 63.08 | 6.818 |
| + $n$-gram match reranker | 69.17 | 7.596 |

Table 7.4: BLEU and NIST scores of different generator setups on the test data.

into training, development, and test data in a 3:1:1 ratio. We use the delexicalized versions of both DAs and natural language paraphrases. We use the three paraphrases as separate instances in the training data, but they serve as three references for a single generated output during validation and evaluation.

We test the three context-aware setups described in Section 7.5 as well as their combinations, and we compare them against the baseline non-context-aware seq2seq generator. Same as for the basic seq2seq model in Chapter 6, we train all our models by minimizing cross-entropy on the training set using the Adam optimizer (Kingma and Ba, 2015), and we measure BLEU score on the development set after each pass over the training data, selecting the best-performing parameters.[15] The content classification reranker is trained in a similar fashion, measuring misclassification on both training and development set after each pass.[16] We repeat the training with five different random initializations of the networks; the automatic scores given in Table 7.4 are averages over the five resulting trained models.

Decoding is run with a beam size of 20 and the penalty weight for content classification reranker set to 100. We set the $n$-gram match reranker weight based on experiments on development data.[17]

| | |
|---|---|
| Context | is there a later option |
| Input DA | iconfirm(alternative=next) |
| Baseline | Next connection. |
| $n$-gram match reranker | You want a later connection. |
| + Prepending context | You want a later connection. |
| + Context encoder | You want a later option. |
| Context | i need to find a bus connection |
| Input DA | inform_no_match(vehicle=bus) |
| Baseline | No bus found, sorry. |
| $n$-gram match reranker | I did not find a bus route. |
| + Prepending context | I'm sorry, I cannot find a bus connection. |
| + Context encoder | I'm sorry, I cannot find a bus connection. |
| Context | i rather take the bus |
| Input DA | inform(vehicle=bus, departure_time=8:01am, direction=Cathedral Parkway, from_stop=Bowling Green, line=M15) |
| Baseline | At 8:01am by bus line M15 from Bowling Green to Cathedral Parkway. |
| $n$-gram match reranker | At 8:01am by bus line M15 from Bowling Green to Cathedral Parkway. |
| + Prepending context | You can take the M15 bus from Bowling Green to Cathedral Parkway at 8:01am. |
| + Context encoder | At 8:01am by bus line M15 from Bowling Green to Cathedral Parkway. |

Table 7.5: Example outputs of the different setups of our generator.

Entrainment is highlighted in the same fashion as in Table 7.4: lexical, syntactic, both.

## Evaluation Using Automatic Metrics

Table 7.4 lists our results on the test data in terms of the BLEU and NIST metrics (see Section 3.6).[18] We can see that while the $n$-gram match reranker brings a BLEU score improvement, using context prepending or separate context

[15]Based on our preliminary experiments on development data, we use embedding size 50, LSTM cell size 128, learning rate 0.0005, and batch size 20. Training is run for at least 50 and up to 1000 passes, with early stopping if the top 10 validation BLEU scores do not change for 100 passes.

[16]We use the same settings except for the number of passes over the training data, which is at least 20 and 100 at most. For validation, development set is given 10 times more importance than the training set.

[17]$w$ is set to 5 when the $n$-gram match reranker is run by itself or combined with the separate encoder, 10 if combined with prepending context.

[18]The outputs are lexicalized using simple text replacement rules before computing BLEU and NIST.

encoder results in scores lower than the baseline.[19] However, using the $n$-gram match reranker together with context prepending or separate context encoder brings significant improvements of about 2.8 BLEU points over the baseline in both cases, better than using the $n$-gram match reranker alone.[20] We believe that adding the context information into the decoder does increase the chances of contextually appropriate outputs appearing on the decoder $k$-best lists, but it also introduces a lot more uncertainty and therefore, the appropriate outputs may not arrive to the top of the list based on decoder scores alone. The $n$-gram match reranker is then able to promote the relevant outputs to the top of the $k$-best list. However, if the generator itself does not have access to context information, the $n$-gram match reranker has a smaller effect as contextually appropriate outputs may not appear on the $k$-best lists at all. A closer look at the generated outputs confirms that entrainment is present in sentences generated by the context-aware setups (see Table 7.5).

In addition to BLEU and NIST scores, we measured the slot error rate ERR (Wen et al., 2015a, see Sections 3.6 and 6.5), i.e., the proportion of missing or superfluous slot placeholders in the delexicalized generated outputs. For all our setups, ERR stayed around 3%.

## Human Evaluation

We evaluated the best-performing setting based on BLEU/NIST scores, i.e., prepending context with $n$-gram match reranker, in a blind pairwise preference test against the baseline (cf. Section 3.6) with untrained judges recruited on the CF crowdsourcing platform. We only used a comparison of our best setting against the baseline to simplify and speed up the annotation process.

The judges were given the context and the system output for the baseline and the context-aware system, and they were asked to pick which of the two system output variants sounds more natural. Ties were not allowed in this ranking setup. We used a random sample of 1,000 pairs of different system outputs over all 5 random initializations of the networks, and collected judgments from three different CF workers for each of them. The judges preferred the context-aware system output in 52.5% cases, significantly more than the baseline.[21]

[19] This is contrary to our experiments on development data, where all three methods brought mild BLEU improvements by themselves.

[20] Statistical significance at 99% level has been assessed using pairwise bootstrap resampling (Koehn, 2004).

[21] The result is statistically significant at 99% level according to the pairwise bootstrap resampling test.

We examined the judgments in more detail and found three probable causes for the rather small difference between the setups. First, both setups' outputs fit the context relatively well in many cases and the judges tend to prefer the overall more frequent variant (e.g., for the context "starting from Park Place", the output "Where do you want to go?" is preferred over "Where are you going to?"). Second, the context-aware setup often selects a shorter response that arguably fits the context perfectly (e.g., "Is there an option at 10:00 am?" is confirmed simply with "At 10:00 am."), but the CF judges seem to prefer the more eloquent baseline variant. And third, both setups occasionally produce non-fluent outputs, which introduces a certain amount of noise.

## 7.7 Discussion

In this section, we compare our context-aware generator to other similar approaches and conclude with a summary and future work ideas.

### Related Approaches

The system presented in this chapter is an evolutionary improvement over the seq2seq system described in Chapter 6 and as such, it is most related in terms of architecture to other recent RNN-based approaches to NLG, which are not context-aware and were already detailed in Section 6.5. In addition, the recent LSTM-based, end-to-end trainable SDS of Wen et al. (2016a) does have an implicit access to previous context while generating responses in a task-oriented dialogue, but the authors do not focus on the influence of context on the generated responses. Our experiments extend the previous approaches by incorporating previous context into response generation and a detailed evaluation of its influence on the outputs.

There have been several attempts at modeling entrainment in dialogue (Brockmann et al., 2005; Reitter et al., 2006; Buschmeier et al., 2010) and even successful implementations of entrainment models in NLG systems for SDSs, where entrainment caused an increase in perceived naturalness of the system responses (Hu et al., 2014) or increased naturalness and task success (Lopes et al., 2013, 2015). However, all of the previous approaches are completely or partially rule-based. Most of them attempt to model entrainment explicitly, focus on specific entrainment phenomena only, and/or require manually selected lists of variant expressions. Our system, on the other hand, learns synonyms and entrainment rules implicitly from the corpus. A direct comparison with previous entrainment-capable NLG systems for SDSs is not possible in our standalone

setting since their rules involve the history of the whole dialogue whereas we focus only on the immediately preceding utterance in our experiments.

## Conclusions and Further Work

We allowed our seq2seq-based generator to exploit preceding context user utterances, thus enabling it to adapt (entrain) to the user's way of speaking and provide more contextually accurate and less repetitive responses. We used two different ways of feeding previous context into the generator and a reranker based on $n$-gram match against the context. Evaluation on our context-aware dataset showed a small but significant BLEU score improvement for the combination of the two approaches, which was confirmed in a subsequent human pairwise preference test.

The entrainment capabilities of the current generator could be further improved by allowing fuzzy matching in the $n$-gram match reranker (e.g., capturing different forms of the same word), experimenting with more ways of incorporating context into the generator, or explicitly controlling the output eloquence and fluency. We reserve these extensions for further work.

# 8

# Generating Czech

This chapter is dedicated to our new, previously unpublished experiments that test the multilingual capabilities of our seq2seq generator from Chapter 6. Unlike most previous works in statistical NLG, we demonstrate these multilingual capabilities in a full-scale experimental setting: We apply our NLG system to Czech, a morphologically rich language very different from English, collecting a new training dataset in order to do so. We also add several extensions to the generation setup in order to tackle problems resulting from Czech rich morphology and to improve output quality.

In the following text, we first describe the motivation for testing our generator experimentally on a different language and for selecting Czech as the target of our study in Section 8.1. Section 8.2 then describes the necessary preparation for this – the creation of a new Czech NLG dataset.

The following three sections then elaborate on the main contribution of this chapter – our practical experiments with Czech seq2seq-based generation. First, in Section 8.3, we present several extensions to the seq2seq generator from Chapter 6 that aim at improving multilingual capabilities of the output for morphologically rich languages. We then describe our experimental setup in Section 8.4 and thoroughly analyze the results obtained in Section 8.5.

As in previous experimental chapters, we include a comparison to related approaches (non-English NLG) and close with a summary of our results and a list of future work plans in Section 8.6.

## 8.1 Motivation

While most current fully statistical NLG systems do not explicitly contain language-specific components and are thus capable of multilingual generation in principle, there has been little work to test these capabilities experimentally. Nearly all previous works on fully trainable NLG known to us focus on English as the target language (see Section 8.6). The relative unavailability of previous multilingual trainable generators goes hand in hand with the scarcity of training datasets. The only available datasets for multilingual NLG known to us are a small sportscasting Korean dataset[1] and several treebanks with detailed deep-syntax/semantic annotation (Hajič et al., 2009), none of which corresponds to our intended usage in a spoken dialogue system.[2]

Since most generators are only tested on English, they do not need to handle grammar complexities that are not present in English but can pose a problem in other languages. A prime example here is the possibility of delexicalized generation (see Section 3.3), present (not only) in all recent RNN-based generators (Wen et al., 2015b,a, and others, see Section 2.3): Generators are trained on delexicalized data with entity names replaced by placeholders to reduce data sparsity. The true entity names are then injected into generator outputs in a postprocessing step. It is typically assumed here that the entity names can be inserted verbatim into the sentence, but this assumption does not hold for languages that require extensive noun inflection.

We therefore apply our generator to Czech to test its practical multilingual capabilities. We selected Czech as a language that we understand and that has two favorable properties for our experiments. First, within the bounds of their shared Indo-European ancestry, Czech is a language very different from English. It has a free word order and very rich inflection, including noun declension with 7 cases and 14 different basic inflection patterns, which are highly ambiguous and subject to further deviations and exceptions (Naughton, 2005, p. 16ff.).[3] This entails a much larger vocabulary and the necessity to address inflection, including the inflection of slot values in delexicalized generation.

[1] Available at `http://www.cs.utexas.edu/users/ml/clamp/sportscasting/` (Accessed: December 17, 2016), cf. Section 2.4.

[2] The Korean sportscasting dataset only contains a limited number of named entities, reducing the need for their inflection (see below). The CoNLL-2009 treebanks are not applicable to the sentence planning stage of NLG, required in SDSs.

[3] Analogous declension patterns are used for adjectives as well as certain numerals and pronouns (Naughton, 2005, p. 48ff.). Verbs have a similarly complex inflection system (Naughton, 2005, p. 131ff.).

Second, Czech is favorable for our experiments since it has a many NLP tools readily available: A part-of-speech and morphological tagger (Straková et al., 2014) and a dependency parser (Novák and Žabokrtský, 2007) are at our disposal. We can also use the Czech rule-based surface realizer from t-trees implemented in the Treex NLP toolkit (Ptáček and Žabokrtský, 2007; Žabokrtský et al., 2008) for experiments with two-step generation (cf. Section 3.4 and Chapter 4). This allows us to easily test and compare various NLG approaches, in a fashion similar to the one employed in Chapter 6.

## 8.2 Creating an NLG Dataset for Czech

Given that no suitable dataset existed for our task and chosen language, we needed to create a new one. Our goal here was to create a dataset comparable in size and domain with existing English NLG datasets for SDSs used in experiments with RNN-based systems. We also aimed at minimizing the duration and costs of collecting the set.

Since there are few to none Czech speakers on crowdsourcing platforms (Pavlick et al., 2014; Dušek et al., 2014), we were not able to use them to recruit a large number of annotators fast and cheap. Recruiting freelance translators seemed easier than recruiting annotators for a specific task and the choice of domain was of little importance to us; therefore, we turned to localizing and translating an existing dataset instead of creating a new one from scratch. We opted for localizing and translating the restaurant dataset of Wen et al. (2015a) due to its convenient size and our familiarity with the domain (see Section 6.5). The dataset contains 5,192 DA-sentence pairs using DAs of 8 different act types (request, confirm, inform, …) and a total of 12 slots describing restaurant properties (name, type, address, area, kids_allowed, …).

In the following, we describe the dataset creation process, starting by a localization of the original English texts, followed by the translation process itself, and finally the necessary postprocessing tasks. We finish this section with a few statistics of the dataset.

### Localizing the Data

We first needed to localize the dataset, replacing the original setting of San Francisco with a Czech one. This is mainly to ensure that the translation results in fluent, natural Czech sentences. In particular, we aimed at using domestic entity names (DA slot values) that need to be inflected since foreign names are often kept uninflected in Czech texts, often using less fluent and rather

***Ananta*** – feminine noun, inflected (nom: *Ananta*, gen: *Ananty*, dat, loc: *Anantě*, acc: *Anantu*, inst: *Anantou*)

***BarBar*** – masculine inanimate noun, inflected (nom, acc: *BarBar*, gen, dat, loc: *BarBaru*, inst: *BarBarem*)

***Café Savoy*** – neuter noun, not inflected

***Místo*** – neuter noun, inflected (nom, acc: *Místo*, gen: *Místa*, dat: *Místu*, loc: *Místě*, inst: *Místem*)

***U Konšelů*** – prepositional phrase, not inflected

Figure 8.1: Examples of restaurant names from the localized data with different morphosyntactic behavior.

Czech noun cases: *nom* = nominative, *gen* = genitive, *dat* = dative, *acc* = accusative, *loc* = locative, *inst* = instrumental (vocative case is not applicable).

conspicuous grammatical constructions to avoid inflection. This is not to say that we avoided using any foreign words in the localization process. Since foreign restaurant names are relatively common the Czech Republic, we also included some of them in the localized data.

We localized the following slots/entities in the dataset: restaurant names, areas/neighborhoods, food types, street addresses, and landmarks. We used a list of randomly chosen restaurant names from the Prague city center as well as lists of Prague neighborhoods, streets, and landmarks. However, we did not aim at creating a real in-domain database of restaurants with their true properties since the veracity of statements in the dataset is irrelevant for our NLG experiments. Therefore, we combined the values in the lists completely at random. The resulting sentences contain contain mostly factually inaccurate, yet meaningful utterances fitting the restaurant information domain.

In addition, the localized lists themselves are relatively short, with just 15 different restaurant names and a similar number of landmarks, streets, and neighborhoods. While much longer lists would be needed for a real-world scenario, the numbers are sufficient to cover most common classes of names that require different inflection patterns or exhibit different syntactic behavior (see Figure 8.1). In addition, it simplifies the task of proof-of-concept generation using lexical information, where values can be used directly (see Section 8.3).

## The Translation Process

If the exact lexicalization is not taken into account, the original dataset contains a lot of duplicate sentences – the total number of DA-sentence pairs in the set is 5,192, but only 2,648 sentences are unique. Therefore, we chose to discard the duplicates from the set for translation, in order to speed up the translation process and lower the costs, albeit at a cost of a lower-quality result.

We recruited six translators and asked them to translate the unique sentences in the set, with random lexicalization as explained above. They were given the following, rather brief, instructions:

- translate the utterances independently of each other,
- strive for fluent Czech, possible to read out aloud,
- do not aim at linguistically precise translation but preserve the meaning, despite factual inaccuracies,
- use varying synonyms (as long as they belong to casual, fluent Czech), including for entity names or slot values (such as price ranges or meal types),
- inflect entity names as needed,
- use formal address (or plural) when addressing the user, and use the female form in the 1st person for self-references.[4]

All rules but the last one are aimed at obtaining a varied and fluent dataset; the last rule then strives for consistency. Note that the translators were not given the input DAs – these carry no more information than the English sentences, and we assume that they would only confuse the translators and hurt the fluency of the results.

The translation process took about a month, and the total cost was CZK 17,000 (ca. $700 in June 2016).

## Translation Postprocessing

After the translation process was finished, three postprocessing steps were performed on the translations: consistency checks, deduplication (to obtain

[4]Czech grammar requires a selection between formal an informal address whenever using a verb in the 2nd person (Naughton, 2005, p. 134ff.). For verbs with past tense or conditional and in any person, gender (or plural) must be selected (Naughton, 2005, p. 140ff.). Here we opted for a feminine form whenever the system addresses itself, and formal address (mostly homonymous with plural) when addressing the user.

the same number of unique delexicalized sentences as in the original set), and expansion (to obtain the same total number of instances as in the original set).

First, we checked the resulting Czech texts for the presence of all required slot values. We took the following iterative, partially automatic approach:

1. Create a list of possible surface forms for all slot values in the dataset, including inflected forms. We used an automatic script and the morphological tagger and generator of Straková et al. (2014) to inflect the surface forms automatically; we checked and corrected errors in the result.

2. Given a DA, check for the presence of surface forms corresponding to its slot values in the corresponding translated sentence. We used an automatic script for this step which prints out all DA/sentence pairs where a slot value may be missing.

3. Given a sentence found by the script to miss a value, check if it contains an alternative surface form not included in the list from Step 1. If so, add this alternative surface form to the list.

4. Otherwise, if the translated sentence does not contain any mention of the DA value, fix the translation.

5. Repeat from Step 2 until there are no missing DA value mentions in the whole set.

Note that these checks result not only in greater consistency of the dataset, but also in a list of possible surface realizations for all slot-value pairs in the dataset. We store this list including morphological information provided by the tagger (with manually corrected errors), and we use this list in our lexicalization experiments (see Section 8.3).[5]

With the dataset contents checked, we performed delexicalization using the surface form list, and we checked for duplicates among the translated sentences to ensure similar variability as in the original English set. If two or more different original English sentences were mapped to the same Czech sentence, we modified the duplicate translations to make them unique, replacing selected words or phrases with synonyms.[6]

[5]We treat even multi-word slot values as single tokens in our surface form list. We assign to them a morphological tag that characterizes the whole multi-word expression best, e.g., a noun tag in nominal groups.

[6]We had to modify 308 translations in total, mostly very short and very frequent sentences, such as those corresponding to the following DAs: bye(), inform(type=restaurant)&inform(name="X-name").

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| *mít* | *pro* | *ty* | *vhodný* | *restaurace* | . | *jeho* | *název* | *být* | *X-name* |
| Mám | pro | Vás | vhodnou | restauraci | . | Její | název | je | Kočár z Vídně |
| I have | for | you | a suitable | restaurant | . | Its | name | is | Kočár z Vídně |

| | | | | | | |
|---|---|---|---|---|---|---|
| *a* | *moci* | *se* | *dát* | *X-food* | *kuchyně* | . |
| a | můžete | si | dát | českou | kuchyni | . |
| and | you can | yourself | give | Czech | cuisine | . |

'I have a suitable restaurant for you. Its name is Kočár z Vídně
and you can have Czech cuisine.'

Figure 8.2: Lemmatized and delexicalized form of the translations for LM scoring.

Example lines from the top: (1) lemmatized and delexicalized Czech used for the LM, (2) original Czech sentence including lexicalization (3) English word-by-word gloss. An English translation is shown below the example.

In the final postprocessing step, we expanded the dataset by re-lexicalizing some of the translations to obtain the same number of instances and the same distribution of different DAs as in the original English set. Given a (delexicalized) DA, a list of corresponding translated sentences, and the total number of corresponding original sentences, we sampled additional copies of the existing translations to match the number of originals. To estimate probabilities of the individual translations for the sampling, we used a 5-gram LM[7] trained on lemmatized and delexicalized translations (see Figure 8.2 for details). We obtained LM scores for all translations corresponding to a DA, used the softmax function to convert the scores into a probability distribution, and sampled additional copies to be re-lexicalized from this distribution. This ensures that translations using overall more frequent words and phrases are more likely to be used multiple times in the set.

We selected random DA slot values for the re-lexicalization, and used surface forms drawn at random from the surface forms list for the given value and roughly corresponding morphology (coarse-grained part of speech for the whole multi-word expression, plus case in nouns/adjectives).[8] Since the morphological information used by this approach is rather crude (e.g., noun/ad-

[7]We used the implementation in the KenLM toolkit (Heafield, 2011). See Section 1.4 for basic information on $n$-gram language models.

[8]We used one part-of-speech tag for nominal and adjectival phrases, one for adverbs and prepositional phrases, and one for verbs. For nominal/adjectival phrases, we detected case for the whole phrase. We were mainly concerned with selecting a syntactically compatible value for re-lexicalization; the correctness of the particular surface form was left for manual checks.

| | English | Czech |
|---|---|---|
| Instances | 5,192 | 5,192 |
| Unique delexicalized instances | 2,648 | 2,752 |
| Unique delexicalized DAs | 245 | 245 |
| Unique lemmas (in delexicalized set) | 399 | 532 |
| Unique word forms (in delexicalized set) | 455 | 962 |
| Average lexicalizations per slot value | 1 | 3.84 |

Table 8.1: Statistics of the translated Czech dataset and a comparison to the English original by Wen et al. (2015a).

The average lexicalizations per slot value shows the number of different surface lexical forms per slot value, as it appears in the dataset. Concrete numeric values (such as street numbers) were disregarded when computing this value. Note that the number of lexicalizations varies widely for different slot types and is approximately inversely proportional to the number of possible values. Also note that if more different restaurant names are used in a real-life scenario, this number will drop. Nevertheless, the number of *possible* lexicalizations, which is even higher (10.70 based on the list of surface forms for our set), would be unaffected.

jective gender is not taken into account), disfluencies ensued in some cases. Therefore, we checked and corrected all re-lexicalized sentences manually, changing inflection or even wording where appropriate. This results in an increased number of unique delexicalized sentences in the set; however, the newly created sentences only show differences in inflection for the most part.

## Dataset Statistics

As described above, the translated set contains the same number of instances as the English original, copies the DA distribution of the original, and contains a slightly higher number of unique delexicalized sentences.[9] A detailed statistics of the dataset size is shown in Table 8.1, along with a direct comparison to the original English set by Wen et al. (2015a). We can see that while the number of unique word lemmas (disregarding restaurant and place names) is slightly higher in the Czech set, the number of unique inflected word forms is more than twice as high. It is also clear that using slot values verbatim in the text is not possible in the Czech set as the number of possible lexical realizations for each value is much higher than one.

[9]Since small changes in wording took place during the final checking, the set has slightly more unique sentences (2,703) even if inflection is not taken into account.

The translated dataset has been released in CSV and JSON formats under the Creative Commons 4.0 BY-SA license. For each instance, it includes the DA as well as the lexicalized and delexicalized surface forms. A JSON list of possible lexicalizations for each slot value including morphological information is also attached. The dataset can be downloaded from the following URL:

`http://hdl.handle.net/11234/1-2123`

A development version of the set is available on GitHub under the following URL:[10]

`https://github.com/UFAL-DSG/cs_restaurant_dataset`

## 8.3 Generator Extensions

We use the seq2seq approach described in Chapter 6 as the base of our experiments in this chapter, consisting of a DA RNN encoder (with DAs represented as lists of triplets "DA type – slot – value") and an attention RNN LM decoder, with an additional reranker penalizing irrelevant or missing information on the output. The following section lists extensions and adjustments to the seq2seq generation process that we implemented to better accommodate generation for Czech.

First, we add a third generator mode to the two-step approach with t-trees and a joint 1-step setup, which are largely unchanged from Chapter 6:[11] lemma-tag generation abstracting away from concrete inflection. We then describe several options for choosing a correct inflected surface form for slot values, which previously have been copied verbatim from the input DA to the output texts. Finally, we introduce the option to generate delexicalized sentences while exploiting lexical information from the input DAs.

### Lemma-Tag Generation

Lemma-tag generation is a third operation mode of our NLG system in addition to the two generator modes described in Chapter 6 (two-step generation separating sentence planning and surface realization and direct one-step generation of surface strings; see a comparison in Figure 8.3). In this mode, an interleaved

[10] At the time of the writing of this thesis, the development version is identical with the release.

[11] For the two-step approach, the simplified t-tree representation has been enhanced with two grammatemes required to capture the meaning: person for personal pronouns and sentence modality (sentence type) for main verbs. Figure 8.3 A) shows interrogative sentence modality (question sentence type) and a 2nd person personal pronoun.

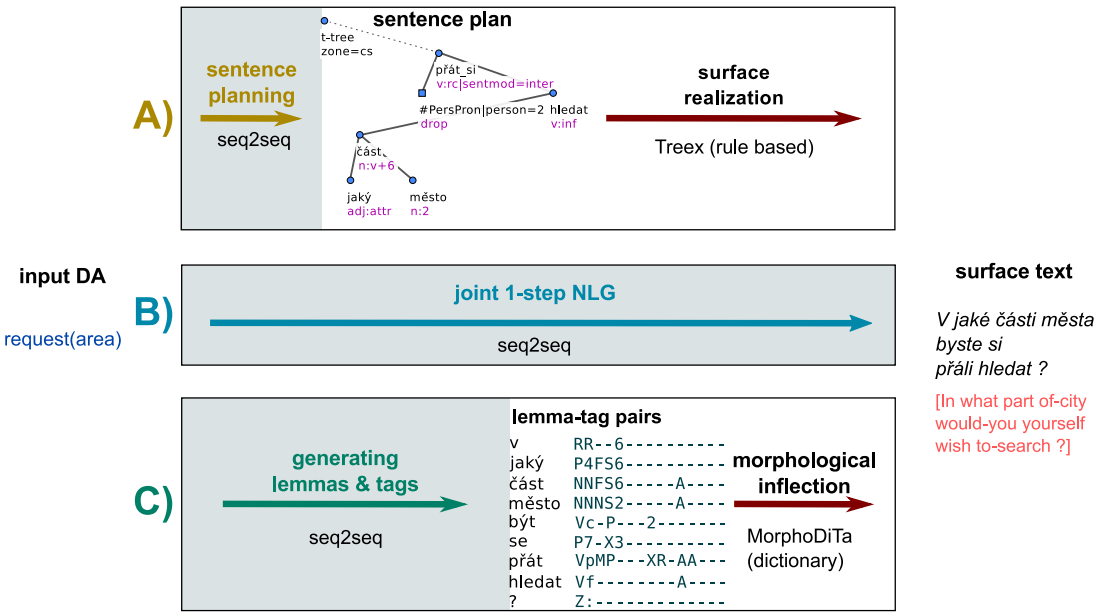


Figure 8.3: A schematic comparison of the three operating modes of our generator.

(A) A traditional 2-step generation with separate sentence planning and surface realization; the seq2seq model produces t-trees (deep syntax trees, see Section 3.5) which are linearized by a mostly rule-based surface realizer (see Chapter 6).[11]

(B) An end-to-end, joint 1-step setup where the seq2seq model directly generates surface strings token-by-token (see Chapter 6).

(C) A newly introduced mode where the seq2seq model produces an interleaved sequence of lemmas and corresponding morphological tags.[12] The lemmas are then inflected into target word forms using the morphological tags and a morphological dictionary.

sequence of lemmas (base word forms) and morphological tags is generated by the seq2seq model (see Figure 8.4). No explicit distinction between lemmas and tags is made in the training data representation for the decoder, except for disjoint dictionaries (cf. Section 6.2); the seq2seq model learns by itself to alternate lemmas and tags regularly on the output. This mode is very close to the joint one-step generation of surface strings – only one of the last surface realization steps, morphological word inflection, is left to a separate module.

Same as the two-step setup with deep syntax trees, the lemma-tag mode does require some training data preprocessing and seq2seq model output post-processing. However, it only amounts to morphological analysis and generation,

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *v* | RR--6---------- | *jaký* | P4FS6---------- | | *část* | NNFS6-----A---- | |
| V | | jaké | | | části | | |
| in | prep+loc | which | pron+int+fem+sg+loc | | part | noun+fem+sg+loc | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *město* | NNNS2-----A---- | *být* | Vc-P---2------- | | *se* | P7-X3---------- | |
| města | | byste | | | si | | |
| city | noun+neut+sg+gen | would you | verb+cond+2nd | | yourself | pron+refl+dat | |

| | | | | | | |
|---|---|---|---|---|---|---|
| *přát* | VpMP---XR-AA--- | *hledat* | Vf--------A---- | *?* | Z:---------- | |
| přáli | | hledat | | ? | | |
| wish | verb+past+pl+anim | to search | verb+inf | ? | punct | |

'In what part of the city would you like to search?'

Figure 8.4: Czech interleaved lemma-tag sequence generated by the new lemma-tag mode of our generator.

From top for each word: (1) generated lemma or tag, (2) the surface word form corresponding to the lemma-tag pair, (3) English gloss. English translation is shown below the sequence.
The Czech positional morphological tags of (Hajič, 2004) are glossed with an abbreviation of part-of-speech and main morphological features: *anim* = masculine animate gender, *2nd* = 2nd person, *cond* = conditional mood, *dat* = dative case, *fem* = feminine gender, *gen* = genitive case, *inf* = infinitive, *int* = interrogative pronoun type, *loc* = locative case, *neut* = neuter gender, *noun* = noun, *past* = past participle, *pl* = plural number, *prep* = preposition, *pron* = pronoun, *punct* = punctuation, *refl* = reflexive pronoun type, *sg* = singular number, *verb* = verb.

a much simpler and relatively standard task where an off-the-shelf external module is available for many languages including Czech. To obtain lemmas and tags for word forms in the training data and to produce inflected word forms from the seq2seq decoding output in our experiments, we use the MorphoDiTa morphological dictionary and tagger (Straková et al., 2014). MorphoDiTa uses the Czech positional morphological tagset of Hajič (2004), which offers a complete coverage of Czech inflection. MorphoDiTa's dictionary covers our domain perfectly except for some slot values (mostly restaurant names) which are not included in the dictionary. Therefore, we use our surface form list compiled when building the dataset (see Section 8.2) for slot value inflection, both in training data analysis and decoded output inflection. We could opt for this combination over using a statistical module such as Flect (see Section 4.4) thanks to a complete coverage of our limited domain. For larger or open domains, a statistical morphology processing module would be required, at least as a backoff for out-of-vocabulary words.

We believe that this approach could partially relieve us of data sparsity issues caused by inflected word forms while it still allows the seq2seq model

to have nearly full control of the nature of the output. Furthermore, we expect this approach to be beneficial to the lexicalization techniques described bellow. Here, the following part-of-speech tag should limit the space of possible surface forms for a slot value. Surface forms not compatible with the part-of-speech tag will be discarded, thus making the task easier.

We will compare the new lemma-tag decoding mode with the two previous ones in our experiments in Section 8.4.

## Advanced Lexicalization

As described in Sections 8.1 and 8.2, Czech requires inflection for most parts-of-speech including proper nouns, which makes delexicalized generation and subsequent lexicalization more complex. Furthermore, our dataset allows not only different inflection forms for a given slot value but also completely distinct, synonymous expressions (analogous to e.g., English *dine* vs. *have dinner*). Therefore, selecting the best lexical forms for a given input DA and delexicalized system output is not trivial, as it was the case in experiments in Chapters 5 through 7: the number of possible surface forms for a given DA slot value is 10.70 on average (see Table 8.1 and Figure 8.5).

We address this problem in two different ways: First, two of our three generator modes (see above) implicitly limit the number of word forms applicable at a given position in the sentence. Second, we introduce four different approaches to selecting one of the applicable word forms. In the following, we describe the applicable word forms limitation, then proceed to the final word form selection.

### Limiting the Number of Applicable Surface Word Forms

The task of selecting a surface form for a certain slot value and position in a delexicalized sentence varies in our three different generation modes (see also Figure 8.5):

**Sentence planning (t-tree generation).** For this generator mode, surface word forms are assigned by the surface realizer, which has been adjusted to prioritize our in-domain surface word form list. However, the sentence planner still needs to select the appropriate lemma as often more lemmas are possible.[13]

The t-tree deep syntax representation contains formemes, a description of the desired morphosyntactic form on the surface (see Section 3.5). Since each slot value placeholder in the generated t-tree has a formeme associated with

[13]For slot values, the t-lemma in the t-tree is always identical to the surface lemma. Therefore, we simply use the term "lemma" here.

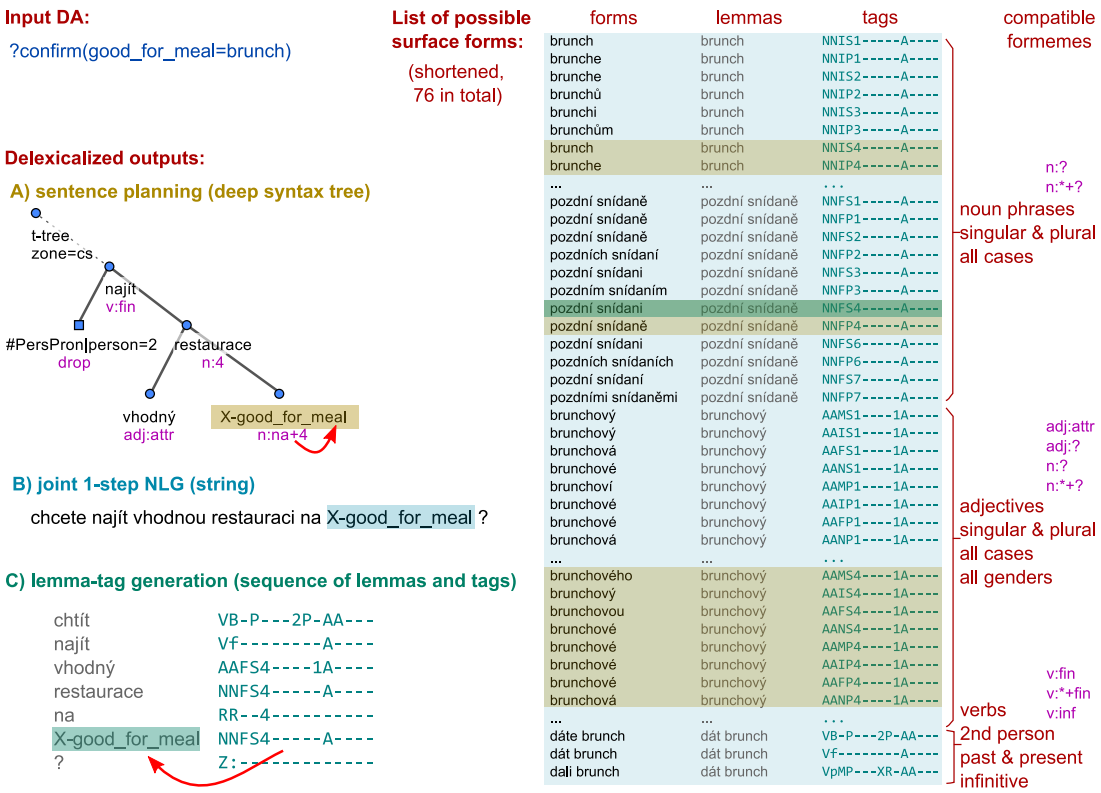


Figure 8.5: The problem of lexicalization for all three generator modes.

Input DA and seq2seq generation outputs in all three modes are shown on the left for the sentence *Chcete najít vhodnou restauraci na brunch?* ('Do you want to find a suitable restaurant for brunch?'). Since the generation is delexicalized, the DA slot value *good_for_meal=brunch* is replaced by the placeholder *X-good_for_meal*.
The task of lexicalization is to select an appropriate surface form for this value from a list of all possible forms, shown on the right. There are many possible surface forms with different parts-of-speech and inflection. Surface forms that are compatible with the outputs generated by the different seq2seq modes are color-coded:

(A) The sentence planning approach selects lemmas at this point; the final word form is then selected in the surface realization phase using the surface forms list. Only lemmas that have surface forms consistent with the formeme are applicable (here: n:na+4).

(B) Joint NLG (direct string generation) has no inherent restrictions on the surface forms; it allows all forms in principle.

(C) The lemma-tag generation only allows forms that have a corresponding part-of-speech tag, including morphological features. In this case, a single form with a perfectly matching tag is available.

Compatible formemes shown in purple on the far right use wildcards: ? stands for any case number (1-7 or X for not inflected), * stands for any preposition or subordinate conjunction.

it, we can limit the set of applicable lemmas only to those that have forms in the surface form list compatible with the given formeme. As the list of surface forms includes main part-of-speech and morphological tags (see Section 8.2), this is a matter of a few simple rules matching formemes with morphological tags. The rules use the syntactic part-of-speech and case in nouns, e.g., the formeme n:na+4 (accusative phrase with the preposition *na* 'onto') allows any noun or adjective phrase in accusative.

If there are no lemmas matching the formeme of the corresponding slot value placeholder in the generated t-tree, no limitation is made and all lemmas are considered applicable.

**Joint 1-step NLG (direct string generation).** In the joint, end-to-end generation mode, no structural phenomena are modeled explicitly; therefore, no distinction can be done among the possible surface forms for a given position in the generated sentence, i.e., all possible surface forms are allowed in all places.

**Lemma-tag generation.** Here, each slot value placeholder in the delexicalized output is followed by the corresponding morphological tag, which can be used to directly limit the possible surface forms: The part-of-speech and a subset of relevant morphological categories for the surface form and the placeholder should match.

For noun and adjective phrases, the relevant categories are gender and case, with a backoff to case only and further to any noun/adjective. For verbs, the only required category is the verb form (infinitive, participle, finite form).[14] Same as with formemes, if no surface form matches the generated tag, no limitation is made and all forms are considered applicable.

#### Selecting the Final Surface Word Form

We implemented four different approaches for the final surface word form selection from a set of applicable forms, two baselines and two language model based ones.

**Random baseline.** The first baseline selects a random applicable word form. This is not suitable for a real application, we only use it to compare against the other approaches.

[14] This is domain-specific. In our domain, all verbal forms on the list use 2nd person polite/plural forms. In other domains, selecting based on person or number could be needed.

**Most frequent baseline.** Here, the applicable word form that occurs overall most frequently in the training data is selected. As can be seen in the experiments in Section 8.4, this very simple approach can be quite efficient in combination with applicable word form limitation described above.

**$n$-gram language model.** We train a 5-gram LM over the output of the seq2seq generator using the KenLM toolkit (Heafield, 2011). Depending on the generator mode, this can be surface word forms (for 1-step joint NLG), serialized interleaved sequences of deep lemmas (t-lemmas) and formemes from a t-tree (for sentence planning with t-trees),[15] or interleaved lemma-tag sequences (for lemma-tag generation). The LM is trained on sequences from training data in the representation corresponding to the generator mode, with slot value placeholders replaced by surface forms as they appear in the original sentences.

The $5$-gram LM is then used to score all applicable surface forms in the context of four generated tokens preceding the slot value placeholder, and the highest-scoring word form is selected for the output.

**RNN-based language model.** The last solution applies exactly the same principle as the previous one, but uses an RNN-based LM instead of an $n$-gram LM (Mikolov et al., 2010, see Sections 1.4 and 2.3). The RNN is composed of LSTM cells operating over the output of the seq2seq generator (same as for the $n$-gram LM), and it is trained to predict a probability distribution for the next token at each step.

Same as with the $n$-gram LM, the RNN LM estimates the probabilities of all applicable surface forms in the given context (the whole sentence preceding the slot value placeholder), and we select the most probable surface form for the output.

## Generating Using Lexical Information in DAs

As shown in Section 8.2, different slot values (proper names) exhibit different morphosyntactic behavior. Moreover, specific slot values also influence the shape of their neighborhood. Some values are more frequently used in specific contexts and some values even postulate a hard requirement for specific word choice or syntactic construction. For instance, certain location names in Czech require the preposition *na* ('on') while other require the preposition *v* ('in'): despite there being no semantic difference, location in the Prague neighborhoods

[15] This is the same sequence representation of t-trees as used in the content classification reranker, see Section 6.2.

of Smíchov and Karlín can only be expressed as *na Smíchově* ('in Smíchov') and *v Karlíně* ('in Karlín'), using the other preposition is unnatural to the native speaker.

In order to capture these subtle differences in the context and to generate appropriate contexts for specific slot values in the input DA, we experiment with lexically-informed generation. Following the recent work of Sharma et al. (2016) and unlike in all previous experiments, where some or all slot values are replaced with placeholders both in DAs and in natural language paraphrases (see Section 3.3), we only use placeholders for the natural language paraphrases here. This allows the generator to access to all the values in the DA and adjust the shape of the output sentence accordingly, but does not require it to produce lexicalized texts, which are only represented sparsely in the training data.

There is no change to the generator algorithm apart from using lexical values in input DAs. In the sequence DA representation (see Section 6.2), we treat all values as atomic; each value has its own embedding. This approach is rather a proof-of-concept one and exploits the relatively low number of different values for each slot in the dataset (see Section 8.2). In a real-world scenario, the values would need to be represented otherwise, such as by their morphosyntactic characteristics (noun gender, required prepositions) or using character embeddings (Luong and Manning, 2016).

## 8.4 Experimental Setup

We experiment on the Czech restaurant information dataset we created in Section 8.2. Following Wen et al. (2015a)'s treatment of the original English data, we split the set into training, development, and test sections in a 3:1:1 ratio.

We use a generator setup very similar to the one applied to English data in Chapters 6 and 7, adding the improvements based on new problems encountered in Czech, as described in Section 8.3. We test all three generator modes therein, combining them with all four lexicalization options as well as with fully delexicalized and lexically-informed generation.

Same as in Chapters 6 and 7, we use the Adam optimizer (Kingma and Ba, 2015) minimizing cross entropy on the training set to train all variants of the main seq2seq generation model as well as the content classification reranker.[16] After each training data pass, we validate the model and keep the best-performing parameters. We use BLEU/NIST scores and classification

[16]The content classification reranker is applied in all setups.

error as the validation criteria for the main seq2seq generator and the reranker, respectively. Based on preliminary experiments on the development set, we keep most parameters at the values used in Section 7.6.[17]

Each of the different lexicalization models (see Section 8.3) requires a different training process:

- The *random baseline* is just using the surface form file provided with the dataset and does not need any additional training.
- For the *most frequent baseline*, training amounts to counting surface form frequencies on the training data.
- The $n$*-gram LM* trains a KenLM 5-gram model with default parameters. All the slot value surface forms are treated as single tokens even if they in fact contain multiple words.
- The *RNN LM* is trained in a very similar fashion as the main seq2seq generator and the content classification reranker, using mostly identical parameters.[18] The target is to minimize cross entropy on predicting the next token; perplexity on the development set is used as the validation criterion (Manning and Schütze, 2000, p. 78). Again, slot value surface forms are always treated as single tokens.

Although multiple data instances often correspond to the same delexicalized DA, we treat each instance individually both in training and testing since the particular lexical values often influence the shape of the whole sentence (see Sections 8.2 and 8.3). This means that only a single reference output is available for each instance in the test set to be used with automatic metrics (see Section 8.5).

In order to avoid unstable performance during validation on the development set, we use all possible paraphrases for the same delexicalized DA found in the training and development sets when measuring the validation BLEU

[17] Both the the main generator and the reranker use embedding size 50, LSTM cell size 128, learning rate 0.005, and batch size 20. At least 50 and up to 1000 training data passes are used for the seq2seq generator, with early stopping if the top 10 validation BLEU scores do not change for 50 passes (this has been changed from 100 to speed up training).

For the reranker, training runs for 100 passes, performance is validated after each pass starting with pass 20, and the best parameters are kept.

Beam size 20 is used for decoding.

[18] We use the same parameters as in the main generator and the reranker (see Footnote 17). The number of passes over the training data is 50, with validation to select the best parameter set performed after each pass starting with pass 10.

score for the main seq2seq generator.[19] Same as in the setup used in Section 6.3, the reranker is validated both on training and development data.[20]

To reduce the effect of random network parameter initialization, we train five networks using different random seeds and use results of all of them for evaluation. In addition, we fix the random seeds so that identical seq2seq generators and rerankers are used in setups that should only differ in the lexicalization method used.

## 8.5 Results

A first look at the outputs confirmed that neither the two-step generation with t-trees nor the lemma-tag generation setup have problems with producing valid sequences that can be postprocessed, i.e., sequences convertible to t-trees or lemma-tag pairs. In fact, based on cursory examination of the outputs, most generated sentences from all setups are understandable Czech, with minor fluency problems and little to no semantic errors (cf. also Table 8.4).

In the following, we first show the results of empirical evaluation using automatic metrics, then describe a human relative ranking experiment with volunteers, and finally attempt to interpret their outcomes by performing a small-scale manual evaluation of our own.

### Evaluation Using Automatic Metrics

Table 8.2 lists the performance of all the different system variants in terms of the BLEU and NIST metrics (Papineni et al., 2002; Doddington, 2002, see Section 3.6). The scores are averaged over all five different random initializations of the networks. The slot error rate ERR (see Section 3.6) stayed below 0.2% for all setups, indicating that semantic errors in the outputs are very rare.

The absolute BLEU/NIST values are much lower than those reported in Sections 6.4 and 7.6; this is largely caused by the fact that the previous experiments had multiple reference paraphrases available. In addition, the larger vocabulary and rich inflection also contribute to overall lower $n$-gram overlap of the generated sentences with the references. This makes BLEU and NIST

[19] Validation BLEU is measured over raw seq2seq outputs, before lexicalization. Using only the single corresponding paraphrase resulted in unstable performance in our preliminary experiments, where we used part of the training data for validation and measured final performance on the development set. Here, a high validation BLEU matching the single reference has often been achieved too early in the training, which resulted in lowered performance when testing.

[20] Classification error on the development set is given 10 times more weight than training set error.

| Setup | | | | |
|---|---|---|---|---|
| **input DAs** | **generator mode** | **lexicalization** | **BLEU** | **NIST** |
| delexicalized | joint (direct to strings) | random | 13.47 | 3.442 |
| | | most frequent | 19.31 | **4.346** |
| | | $n$-gram LM | 19.40 | 4.274 |
| | | RNN LM | **19.54** | 4.273 |
| | lemma-tag | random | 17.18 | 3.985 |
| | | most frequent | 18.22 | 4.162 |
| | | $n$-gram LM | 17.95 | 4.132 |
| | | RNN LM | 18.51 | 4.162 |
| | two-step with t-trees | random | 14.93 | 3.784 |
| | | most frequent | 16.16 | 3.969 |
| | | $n$-gram LM | 16.13 | 3.970 |
| | | RNN LM | 16.39 | 3.974 |
| lexically informed | joint (direct to strings) | random | 12.56 | 3.300 |
| | | most frequent | 17.82 | 4.164 |
| | | $n$-gram LM | 17.85 | 4.082 |
| | | RNN LM | 17.93 | 4.094 |
| | lemma-tag | random | 19.96 | 4.306 |
| | | most frequent | 20.86 | 4.427 |
| | | $n$-gram LM | 20.54 | 4.399 |
| | | RNN LM | **21.18** | **4.448** |
| | two-step with t-trees | random | 16.13 | 3.919 |
| | | most frequent | 17.15 | 4.073 |
| | | $n$-gram LM | 17.24 | 4.078 |
| | | RNN LM | 17.62 | 4.112 |

Table 8.2: Performance of the different generator setups in terms BLEU and NIST on our test set.

Best results for delexicalized and lexically informed generation are marked in bold (significant difference against all other results in the same category has been assessed using paired bootstrap resampling (Koehn, 2004) at 95% confidence level).

less reliable, but they still keep their validity, given the limited domain and the size of the test set. Our cursory manual checks of the outputs indicated that they reflect the output quality quite well.

If we compare the individual scores, we can see that the lexically-informed setup tends to produce better results with lemma-tag generation and the tree-based setup, but it performs worse in direct string generation. A possible cause for this discrepancy is the fact that the lexically-aware setup deals with more diverse inputs, and combined with the diversity of the direct string generation output, where all possible inflection forms must be generated by the seq2seq model, this makes data sparsity too large, rendering the generator unreliable.

The generator modes show a similar picture as in Chapter 6: the joint, direct string generation performs better than the tree-based setup, despite the complex Czech surface morphology. The lemma-tag generator mode ranks comparably to direct string generation, outperforming it in the lexically informed setting while staying behind in delexicalized generation. We believe that the lemma-tag setup in the lexicalized setting is probably able to better utilize the extra information about slot values; however, a partial explanation to this could also be given by a better match against the references since the result is not so pronounced from a cursory reading of lemma-tag generation outputs in both settings (both outputs only show minor, scattered disfluencies).

As to the lexicalization methods, the low performance of the random baseline only demonstrates that attention must be paid to selecting the correct surface form for a slot value. The most frequent baseline, however, performs relatively well when compared with more sophisticated methods. The $n$-gram LM is only able to outperform it by a narrow margin in direct string generation, working directly over surface tokens, and in t-tree-based generation in lexically-informed mode. For lemma-tag generation and delexicalized t-tree-based generation, the most frequent baseline works better than the $n$-gram LM. This suggests that the basic $n$-gram LM is less well suited for usage over interleaved sequences of non-surface tokens (lemmas/t-lemmas and tags/formemes); using a factored LM (Bilmes and Kirchhoff, 2003) should improve the performance. Another problem with interleaved sequences is the fact that the LM works effectively over a shorter span of the sentence, with five tokens representing only three consecutive (t-)lemmas and two tags or formemes.[21] Increasing the order of the model for these settings could therefore also improve the prediction. The RNN LM is consistently superior to both the baselines and the $n$-gram LM, supporting the trend observed in the standard language mod-

[21] This problem seems to be more pronounced in lemma-tag generation than in t-tree-based generation, where a single t-tree node often represents multiple surface sentence tokens.


Figure 8.6: Human preference ranking using the Appraise system.

Top left: the input DA (formatted for easier human reading). Top right: the reference sentence. Bottom: system outputs to be ranked.
The instruction text just above the list of outputs reads: *Please sort the generated sentences below from best to worst (with regard to the reference above). You can assign the same rank to multiple sentences.* The colored buttons read: *Best, Rank 1, …, Rank 5, Worst*.
The rankers' task is to click the rank buttons according to their preferences; the sentences reorder automatically according to the ranks. Ties and non-continuous ranks are allowed (as shown in the picture) since the ranks are later converted to multiple binary comparisons between pairs of systems.

eling task (Mikolov et al., 2010; Kim et al., 2015). Part of its success is probably given by its ability to exploit longer contexts than the $n$-gram LM.

The overall best results in terms of BLEU were achieved with the RNN LM lexicalization, combined with direct seq2seq generation to strings in the fully delexicalized setting and with lemma-tag generation in the lexically-informed setup. The most frequent baseline in the delexicalized setting shows better NIST results, but the difference is very small. A manual cursory check of the outputs came out in favor of the RNN-based setup.

| Setup | | | True | Order |
|---|---|---|---|---|
| input DAs | generator mode | lexicalization | Skill | |
| delexicalized | joint (direct to strings) | RNN LM | 0.511 | 1 |
| delexicalized | lemma-tag | RNN LM | 0.479 | 2-4 |
| lexically informed | lemma-tag | RNN LM | 0.464 | 2-4$*$ |
| lexically informed | lemma-tag | most frequent | 0.462 | 2-4 |
| lexically informed | joint (direct to strings) | RNN LM | 0.413 | 5 |
| lexically informed | two-step with t-trees | RNN LM | 0.343 | 6-7 |
| lexically informed | lemma-tag | $n$-gram LM | 0.329 | 6-7 |

Table 8.3: Results of the human subjective preference rankings.

The table lists all the setups selected for human preference ranking, along with their performance as computed by the TrueSkill algorithm (Sakaguchi et al., 2014, see text). Bootstrap resampling with 1,000 samples and 95% confidence level is used to establish clusters of systems with significantly different performance; this is shown in the "Order" column and by the dotted lines.
"$*$" marks the best system in terms of automatic metrics.

## Human Relative Preference Rankings

Similarly to Section 7.6 and Wen et al. (2015a), we performed a subjective human preference ranking experiment to complement the automatic scores. However, we made two significant changes to the experimental setup used in Section 7.6:

1. Since there is a large number of system variants and no valid criterion which would preselect a single pair of systems, we needed to compare multiple system variants. This would result in an excessive number of pairwise comparisons being required. Therefore, we opted for a multi-way ranking of system outputs, as used in several past WMT MT system evaluation campaigns (Bojar et al., 2014, 2015, 2016c).[22]

2. Due to the lack of Czech speakers on crowdsourcing platforms (see Section 8.2), we needed to ask volunteers among friends and colleagues for ratings. This is also common practice in WMT evaluations (cf. e.g., Bojar et al., 2016c).

Since we expected to only be able to obtain a limited number of ratings, we still needed to preselect only a limited number of system variants for human evaluation. We based our selection on the best system in terms of BLEU

[22]Ranking outputs of multiple system variants for the same input together is frequent in NLG evaluations, too; see Section 3.6 for details.

(lexically informed, lemma-tag, with RNN lexicalizer), and selected a list of contrastive systems, changing just one of the parameters. This lead to a total of 6 systems (one additional for fully delexicalized generation, two additional for generator mode, and two additional for different lexicalizers, disregarding the clearly inferior random lexicalization). In addition, we included the best delexicalized configuration (direct string generation with RNN lexicalizer). The list of compared systems is shown in Table 8.3.

In addition, we filtered the system outputs to be ranked, leaving out „trivial" sentences (sentences which did not contain any slot value, such as greetings or requests). Cursory manual checks indicated that output quality in these sentences is generally near-perfect, and that the particular wording mostly depends on random network initialization. We used outputs from all five system runs with different random initializations.

The ranking experiment has been performed using a slightly modified version of the Appraise MT evaluation software (Federmann, 2012).[23] The experiment participants were given the DA (in a human-readable form), a reference sentence, and up to five anonymized system outputs to rank (see Figure 8.6). Same as in WMT evaluations (cf. Section 3.6), the human rankers were only instructed to look at the DA and the reference and to order the system outputs from "best" to "worst", according to their own subjective criteria. The rankings are then converted to pairwise comparisons during further analysis. Ties are allowed as sometimes it is not possible to properly distinguish among some of the sentences. Discontinuous system ranks (e.g., 1-1-2-4-4 as in Figure 8.6, missing rank 3) are allowed since this does not affect the pairwise judgments.

We used Appraise to randomly sample instances for humans to rank, consisting of a DA, a corresponding reference and a subset of corresponding system outputs (with maximum five outputs). Instances where multiple systems produce the same sentence are automatically merged by Appraise before creating the human ranking instances; they are unfolded after rankings when pairwise comparisons are produced.

We were able to get a total of 1,125 rankings from 73 different human rankers (with the most prolific ranker submitting 69 rankings, and 18 rankers in total producing more than 30 rankings each). These rankings unfolded to a total of 15,671 pairwise system comparisons, which is comparable to the number of comparisons elicited in WMT evaluation campaigns (Bojar et al., 2016c).

[23]The original version can be found at `https://github.com/cfedermann/Appraise` (Accessed: Feb 25, 2017). The code for our modified version is accessible at `https://github.com/tuetschek/Appraise/tree/cs_rest`. Our modifications are solely related to the user interface, not the overall organization of the task.

The pairwise comparisons were evaluated using the TrueSkill algorithm (Herbrich et al., 2006) as applied in the recent WMT evaluations (Sakaguchi et al., 2014; Bojar et al., 2015, 2016c). TrueSkill models the performance of each system $S_i$ using a Gaussian distribution with a mean $\mu_{S_i}$ (the system skill) and variance $\sigma^2_{S_i}$ (representing the uncertainty about system $S_i$'s performance). The pairwise comparisons are then used to gradually update each system's skill estimate; the size of the update is determined by the surprise coming from the particular comparison (update is bigger if a low-ranking system beats a high-ranking one) as well as the current skill estimate accuracy (updates are lower for a low variance $\sigma^2_{S_i}$). The WMT campaigns use 1,000 bootstrap-resampled TrueSkill estimates to establish statistically significant differences at 95% confidence level, effectively producing clusters of similarly-performing systems; we followed this practice here.

The results of TrueSkill applied to our human rankings are shown in Table 8.3. We can see that four clusters in total were produced by the algorithm, establishing direct generation to strings from delexicalized DAs with RNN LM lexicalization as the best system. This system was the best among delexicalized setups in terms of BLEU, but not the best overall in terms of automatic metrics: The setup that achieved the highest BLEU and NIST (lexically informed lemma-tag generation with RNN LM lexicalizer) ranked third and was assigned to the second cluster with two other, similarly performing systems.

In addition to system rankings in the Appraise tool, we asked the raters for informal, qualitative comments (over Facebook or email) on the systems' performance and the rating task itself, and we received comments of widely varying length from 11 participants in total. Apart from general discussion about the system performance and/or particular errors in the system outputs (and even the references in isolated cases), two important issues were raised in the comments:

- The lack of guidelines for rating did not feel convenient to some participants, and they required further instructions (e.g., on how to weigh semantic against fluency errors).

- The systems reflect the high diversity present in the training data, producing often very different outputs for a certain DA. While synonymous variants are presented as equal to the systems during training, some participants did not prefer certain variants, i.e., complained about them as errors. Two particular cases occurred in our experiment:

– There are a lot of paraphrases for the DA inform(name=X-name, type=restaurant). Some of them plainly state that *X-name is a restaurant*, while others praise the restaurant, e.g., *I know a very pleasant place called X-name.* Participants complained about the latter as fabricating information not present in the DA.
– Another problem concerned DAs with the slot kids_allowed. Here, the main problem was proper localization – due to anti-discriminatory laws, restaurants in the Czech Republic cannot disallow entrance with children without further reasons. Therefore, the participants rejected paraphrases stating outright that *There is no entry allowed for children*, and preferred milder variants, such as *The restaurant is not appropriate for children*.

Although weighing different error types may be problematic to some users, it has been shown to produce more consistent rankings than rating along separate scales (Callison-Burch et al., 2007; cf. Section 3.6). We believe that this does not present a problem to the quality of the ratings; instead, a better explanation should be provided to the users, such as intended usage examples for the system.[24]

The high training data diversity represents a double-edged blade. On one hand, it leads to diverse system outputs, which is a positive feature. On the other hand, the fact that the selection of a certain variant can only result from random events occurring during system training, such as NN initialization or training example order, presents a certain amount of noise in the ratings, be it with automatic metrics (where some variants may be better covered by references) or human ratings (where some variants may be preferred). The system rated as the best by users is indeed the most preferred one, but not always thanks to the particular configuration.

The discrepancy in system results for automatic metrics and human ratings and the noise problem observed during human evaluation lead us to a closer examination of the best systems' outputs.

### Manual Analysis of a Sample of the Best Systems' Results

We performed a small-scale evaluation on a sample of the two best systems' outputs – the best BLEU and NIST-rated system (lexically informed lemma-tag generation with RNN LM lexicalizer) and the most preferred system selected in the human ratings (fully delexicalized direct string generation with RNN LM

[24]This was found helpful by one of the participants asking for more instructions.

| | |
|---|---|
| Input DA | inform(name="Švejk Restaurant", near=Stromovka, price_range=cheap) |
| Reference | *Restaurace Švejk je poblíž Stromovky a nabízí nízké ceny.*<br>Švejk Restaurant is near Stromovka and it offers low prices. |
| Best BLEU/NIST | *Restaurace Švejk je levná restaurace, poblíž Stromovky.*<br>Švejk Restaurant is a cheap$_{\textbf{fem}}$ restaurant$_{\textbf{fem}}$, near Stromovka. |
| Most preferred | *Restaurace Švejk je levná podnik blízko Stromovky*<br>Švejk Restaurant is a cheap$_{\textbf{fem}}$ place$_{\textbf{inan}}$ close to Stromovka |
| Input DA | inform(name="U Konšelů", phone=X-phone) |
| Reference | *Do U Konšelů můžete zavolat na telefonu 242817033.*<br>to U Konšelů you can call on the phone 242817033. |
| Best BLEU/NIST | *Cenu do restaurace U Konšelů můžete volat na číslo 242817033.*<br>price to the restaurant U Konšelů you can call the number 242817033. |
| Most preferred | *Telefonní číslo U Konšelů je 242817033.*<br>the phone number of U Konšelů is 242817033. |
| Input DA | inform(type=restaurant, good_for_meal=dont_care, count=3) |
| Reference | *Jsou zde 3 restaurace, které jsou vhodné pro jakékoli jídlo*<br>there are 3 restaurants$_{\textbf{nom}}$ that are suitable for any meal |
| Best BLEU/NIST | *Našla jsem 3 restaurací, které jsou vhodné pro jakékoli jídlo.*<br>I found 3 restaurants$_{\textbf{gen}}$, which are suitable for any meal. |
| Most preferred | *V nabídce je 3 restaurací, které nabízí všechny druhy jídel.*<br>on the list are 3 restaurants$_{\textbf{gen}}$, that offer all kinds of meals. |
| Input DA | inform_no_match(area=Vinohrady) |
| Reference | *Lituji, ale v lokalitě Vinohrad se nenachází žádné restaurace.*<br>I am sorry but in the area of Vinohrady are not located any restaurants |
| Best BLEU/NIST | *Na Vinohradech nejsou žádné restaurace vhodné pro děti.*<br>in Vinohrady there are no restaurants appropriate for children. |
| Most preferred | *V Vinohrad nejsou žádné takové restaurace.*<br>in Vinohrady there are no such restaurants. |
| Input DA | inform(area=Hradčany, type=restaurant, kids_allowed=no, count=2) |
| Reference | *V lokalitě Hradčan jsem našla 2 restaurace, které nedovolují vstup dětem.*<br>in the area of Hradčany I found 2 restaurants, which do not allow entry to children. |
| Best BLEU/NIST | *V oblasti Hradčan se nabízí 2 restaurace, které nejsou vhodné pro děti.*<br>in the area of Hradčany offer themselves 2 restaurants, which are not appropriate for children. |
| Most preferred | *Na Hradčany se nehodí 2 restaurace, které nejsou vhodné pro děti.*<br>for Hradčany are not suitable 2 restaurants, which are not appropriate for children. |

Table 8.4: Examples of outputs generated by the best two configurations (a selection of erroneous outputs, with errors highlighted).

Errors are color-coded: fluency (inappropriate word added), lexicalization (feminine adjective form used with a masculine inanimate noun), inappropriate structure (3rd example: the genitive *restaurací* is not appropriate for numerals lower than five; 4th example: the preposition *v* is not suitable for the lexical value *Vinohrady*), semantic errors (4th example: irrelevant information added; 5th example: meaning shifted). Subscripts indicate morphological categories (*fem* = feminine gender, *gen* = genitive case, *inan* = masculine inanimate gender, *nom* = nominative case).

| Setup | Errors of type | | | | | Errors |
|---|---|---|---|---|---|---|
| | F | L | IS | P | S | Total |
| Best BLEU/NIST (lex. inf., lemma-tag, RNN LM) | 9 | 7 | 2 | 2 | 2 | 22 |
| Most preferred (delex., joint/direct, RNN LM) | 5 | 6 | 4 | 8 | 1 | 24 |

Table 8.5: Number of errors of various types found during our manual analysis

Error types: F = fluency (except in lexicalization), L = lexicalization (selecting wrong surface form), IS = inappropriate structure (sentence not appropriate for a particular slot value), P = missing final punctuation, S = semantic errors.

lexicalizer). We randomly selected 100 sentences from the outputs over all five random initializations and manually assessed errors of different types (cf. also Table 8.4):

- *Fluency* – any problems concerning the fluency of the sentence, apart from wrong inflection of slot value surface forms (i.e., fluency problems incurred in the main seq2seq generator or its postprocessing stage).
- *Lexicalization* – fluency problems arising from the selection of an inappropriate surface form for a slot value in the lexicalizer, typically a wrong inflection form.
- *Inadequate structure* – cases where a particular slot value required a certain shape of its surroundings (see Section 8.3), which was not reflected by the main seq2seq generator. The lexically-informed setup should avoid such errors.
- *Punctuation* – cases where final sentence punctuation has been omitted.
- *Semantic* – cases where the generated sentence does not fully reflect the meaning of the input DA due to added, modified, or missing information.

We did not judge the particular choice of synonymous output variants, as far as they were fluent and adequate for the current DA.

Our analysis has shown that semantic errors are indeed rare in both setups' outputs. We found one case where the most preferred setup mixed the meaning of two clauses and two instances where the best BLEU/NIST setup added information not present on the input (see Table 8.4). None of these cases could be revealed by the automatic ERR metric.

Fluency problems are more frequent as expected, and they occur in both setups. A common problem in lexicalization is selecting the correct inflection

form for adjectives, which is typically determined by the following noun (see Table 8.4). This shows the limits of the applied RNN LM lexicalizer, which only takes previous words into account. Applying the RNN LM in both directions thus should improve performance. Problems with inadequate structure for a particular slot value occur more rarely than we expected. There are four cases in the delexicalized (most preferred) setup and two cases for the lexically informed one (best BLEU/NIST). The lexically informed generator can indeed create compatible surrounding for some surface forms, but was not able to handle the complex syntactic behavior of Czech numerals (see Table 8.4).[25]

The total numbers of all types of errors found during our analysis can be seen in Table 8.5. They show that both setups perform very comparably. We can see that while the best setup in terms of BLEU/NIST made slightly less errors in total, the errors made by the most preferred setup often only concern punctuation. If we disregard punctuation, the most preferred setup seems to fare slightly better.

## 8.6 Discussion

In the final section of this chapter, we first summarize other approaches to non-English NLG, then offer a few concluding remarks.

### Comparison to Related Approaches

As already noted in Section 8.1, NLG experiments for non-English languages are relatively rare and fully trainable approaches even rarer. Our work is, to our knowledge, the first application of neural NLG to a non-English language for data-to-text generation.

Most works concerned with multiple languages focus on surface realization. There have been a few approaches using handcrafted grammars with a varying level of coverage for different languages (Bateman, 1997; Allman et al., 2012). The SimpleNLG realizer of Gatt and Reiter (2009), which takes a procedural approach, has also been ported into several languages (Bollmann, 2011; Vaudry and Lapalme, 2013; Mazzei et al., 2016). Further works using multilingual rule-based surface realization pipelines were developed in the context of machine translation (Aikawa et al., 2001; Žabokrtský et al., 2008; Dušek et al., 2015; see Chapter 4). Bohnet et al. (2010) created the only fully statistical multilingual

[25]The Czech numerals *one*, *two*, *three*, and *four* behave like adjectives, with their governing noun inflected according to its position in the sentence, while higher numerals behave like nouns; the quantified noun then takes on the genitive case (Naughton, 2005, p. 113ff.).

realizer known to us, which is based on a pipeline of SVMs. They perform their experiments on English as well as German, Spanish, and Japanese.

The work of Chen et al. (2010) and the follow-up of Kim and Mooney (2010) is the only non-neural NLG work with multilingual experiments known to us which uses a joint setup with sentence planning and surface realization in a single model. They generate English and Korean sportscasting sentences using an inverted semantic parser (see Section 2.3).

There have been works on neural non-English NLG, but they are so far limited to the very specific task of Chinese poetry generation. Zhang and Lapata (2014) use a modified RNN LM while Yi et al. (2016) and Wang et al. (2016) apply a modified seq2seq encoder-decoder architecture. All of these works are mostly concerned with the form of the generated poems (meter, rhymes), whereas we also concentrate on the semantic content of the generated outputs. Another line of work in multilingual neural NLG is only concerned with morphological inflection (see Section 4.4).

Also related to our work, and perhaps most inspiring for our further work, are the recent neural MT experiments specifically targeting the problems of rich morphology within the seq2seq framework. Luong and Manning (2016) present a hybrid model that uses word-based generation for frequent words but generates rare words character-by-character. Sennrich et al. (2016) apply standard seq2seq MT models over words split into sub-word units using an adapted version of the byte pair encoding compression algorithm, where the length of sub-word units is inversely proportional to their frequency. Both approaches are able to improve upon the basic seq2seq word-to-word translation.

## Conclusions and Future Work

We applied our seq2seq generator from Chapter 6 to a different language, Czech, and we extended it in several ways to address the much richer morphology, including the need to inflect slot values, such as restaurant names, which are simply copied verbatim from the DA in all previous neural NLG systems, including our experiments in Chapters 6 and 7. We added the possibility to generate into interleaved sequences of lemmas (base word forms) and morphological tags, which are then postprocessed by a morphological generator. We also implemented two baselines and two LM-based methods of selecting appropriate inflected surface forms for slot values.

The results in Section 8.5 show that our generator is able to deal with richer Czech vocabulary and surface form inflection in most configurations, and produces mostly relevant and fluent outputs. This demonstrates that the size

of our dataset, comparable to previous experiments on English, is sufficient to train the generator for Czech.

The results came out largely in favor of the simplest generator setups, especially in the case of manual evaluation. They suggest that using two-step generation with t-trees, where the seq2seq generator works as a sentence planner and its outputs are postprocessed using a standalone surface realizer, leads to worse results than direct generation into surface strings. Generating sequences of lemmas and tags performs comparably to direct string generation (but slightly worse in the user preference test); a direct generation setup is thus preferable thanks to its simplicity. Experiments with generating delexicalized outputs from lexicalized input DAs did not bring any performance improvements; the additional information from lexicalized DAs is probably too sparse to be useful to the network.

On the other hand, a sophisticated slot value surface form selection clearly pays off. Our RNN LM model consistently performed better than both baselines and the $n$-gram LM. Our manual analysis showed that there is still some room for improvement by using bidirectional RNN LMs.

For future work, apart from applying a bidirectional RNN LM for surface form selection, we will investigate the possibilities of removing the need for prescribed surface form lists altogether. We plan on using character-based morphological generation (Kann and Schütze, 2016; Luong and Manning, 2016, see Section 4.4) for the surface forms, and we will also investigate the possibility of using sub-word units (Sennrich et al., 2016) in the whole generation setup to reduce vocabulary sparsity.

# 9 Conclusions

This chapter gives the final brief overview of the results of our experiments, organizing them along the main objectives of the thesis, and provides a short outlook on direct extensions of our work as well as long-term future work ideas.

The main contributions of our thesis addressing the individual objectives preset in Chapter 1 are as follows:

**A) Generator easily adaptable for different domains.** In Chapter 5, we developed an A*-search-based NLG system that is trainable from pairs of natural language sentences and corresponding dialogue acts, without the need for fine-grained semantic alignments, thus greatly simplifying training data collection for NLG. It was the first NLG system to learn alignments jointly with sentence planning. This system has then been superseded by a new, seq2seq-based one in terms of both speed and output quality, as described in Chapter 6. The seq2seq-based system reached new state-of-the-art without fine-grained alignments on the small BAGEL dataset (Mairesse et al., 2010), using much less training data than other RNN-based approaches. The two NLG systems were described in (Dušek and Jurčíček, 2015) and (Dušek and Jurčíček, 2016b), respectively.

**B) Generator easily adaptable to different languages.** We developed a simple, domain-independent surface realizer from the t-trees deep syntax formalism (see Section 3.5) for English to use with with our A*-search-based generator, similar to an older Czech realizer used in the Treex/TectoMT NLP framework (Popel and Žabokrtský, 2010). We have simplified the creation of t-tree realizers for new languages by code reorganization and, more importantly, by creating a novel, fully statistical morphological inflection module, which is trainable from

relatively small amounts of morphologically annotated data and generalizes to previously unseen word forms (see Chapter 4). Our morphological generator as well as parts of the English realizer code have since been reused by the QTLeap project for deep syntactic machine translation in several other languages (Popel et al., 2015a; Aranberri et al., 2016). The English realizer was described in (Dušek et al., 2015), and we reported on the morphological inflection module in (Dušek and Jurčíček, 2013).

In Chapter 8, we applied our seq2seq-based generator to Czech, a language with much richer morphology and freer word order than English. We experimented with several novel approaches to generating text in a morphologically rich language, addressing larger vocabulary and the need to inflect proper names (DA slot values), which is not needed in English. We show that our seq2seq-based generator is able to produce mostly correct and fluent sentence structures without any significant changes, apart from proper name inflection, where our RNN LM based module significantly outperforms a strong baseline.

**C) Generator that adapts to the user.** Mimicking human behavior in dialogue, where interlocutors entrain (adapt their wording and syntax) to each other, we extended our seq2seq generator in Chapter 7 to reflect not only the input DA, but also the wording and syntax of the previous user request, thus enabling it to create responses appropriate in the preceding dialogue context and providing it with a natural source of variation. The context-aware generator achieved a small but statistically significant performance improvement over the context-oblivious baseline. This result has been described in (Dušek and Jurčíček, 2016c).

**D) Comparing different NLG system architectures.** In Chapters 6 and 8, we compare two different NLG architectures: a two-step pipeline using separate sentence planning and surface realization modules and a joint setup generating surface strings directly. Thanks to the flexibility of the seq2seq approach, we are able to use essentially the same model for both generation setups, generating t-trees (deep syntax postprocessed by a surface realizer) or surface word forms (in an end-to-end fashion). Furthermore, to simplify the problem of rich morphological inflection in Czech in Chapter 8, we experiment with seq2seq generation of lemma-tag sequences (base word forms and morphological categories), which are subsequently postprocessed by a morphological generator.

In our experiments, seq2seq models learn to generate valid t-trees and lemma-tag sequences successfully. However, the direct, end-to-end setup gen-

erating surface word forms reaches superior performance. This shows that for our domains, abstracting away from surface grammar phenomena does not pose an advantage big enough to outweigh possible error accumulation in a more complex pipeline setup; not even for Czech with its complex morphology. Experiments for English from Chapter 6 were described in (Dušek and Jurčíček, 2016b).

**E) Dataset availability for NLG in SDSs.** To perform our experiments in Chapters 7 and 8, we have created two novel datasets for NLG, which are freely available under a permissive license:[1] the first NLG dataset for Czech, which is simultaneously the biggest freely available non-English NLG dataset, and the first NLG dataset using preceding dialogue context and specifically targeted at adapting system responses to the user. The latter set is also described in (Dušek and Jurčíček, 2016a).

In sum, our work constitutes significant advances along all of the preset objectives. In a few aspects, it leaves room for improvement in future work as some of the experiments on dialogue alignment and Czech generation were rather limited. Nevertheless, the generator we implemented is fully functional and usable in practice, within a spoken dialogue system or in a standalone setting. It is freely available for download from GitHub at the following URL:[2]

`https://github.com/UFAL-DSG/tgen`

As direct extensions of our work, we would like to widen the user adaptation experiment by taking the whole dialogue into account. We also plan to remove the need for manual inflected proper names lists in Czech generation by using a character-based seq2seq morphological generation module (such as Kann and Schütze, 2016).

In the future, we will work on obviating the need for delexicalizing proper names in NLG altogether in order to further simplify portability of NLG systems to other domains and languages. We are considering using input copying methods for seq2seq setups (Gu et al., 2016; Lebret et al., 2016) as well sub-word unit generation (Sennrich et al., 2016). We are also interested in NLG for broader domains (Lebret et al., 2016) and simplifying the cross-domain portability of trained models (Wen et al., 2016c). Finally, we see the future of NLG ultimately

[1] Available for download at `https://github.com/UFAL-DSG/alex_context_nlg_dataset`, `https://github.com/UFAL-DSG/cs_restaurant_dataset` under the Creative Commons 4.0 BY-SA license.

[2] The generator is distributed under the Apache 2.0 license.

in end-to-end solutions incorporating language understanding, dialogue management, and response generation. These end-to-end systems, which are now mostly limited to non-task-oriented settings without any knowledge grounding (e.g., Vinyals and Le, 2015; Serban et al., 2016), although proof-of-concept experiments for task-based systems are also starting to appear (Wen et al., 2016a; Eric and Manning, 2017; Williams et al., 2017; Ghazvininejad et al., 2017), have the potential to overtake the whole field of spoken dialogue systems and natural language human-computer interaction in general.

# Bibliography


Abadi, M. et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015. Available at: `http://tensorflow.org/`.

Ahlberg, M. – Forsberg, M. – Hulden, M. Semi-supervised learning of morphological paradigms and lexicons. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, p. 569–578, Gothenburg, Sweden, 2014.

Ahlberg, M. – Forsberg, M. – Hulden, M. Paradigm classification in supervised learning of morphology. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, p. 1024–1029, Denver, CO, USA, 2015.

Aikawa, T. – Melero, M. – Schwartz, L. – Wu, A. Generation for multilingual MT. In *Proceedings of the MT-Summit*, p. 9–14, Santiago de Compostela, Spain, 2001.

Allman, T. – Beale, S. – Denton, R. Linguist's Assistant: A Multi-Lingual Natural Language Generator based on Linguistic Universals, Typologies, and Primitives. In *INLG 2012 Proceedings of the Seventh International Natural Language Generation Conference*, p. 59–66, Utica, IL, USA, 2012.

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

Aranberri, N. – Labaka Intxauspe, G. – Jauregi, O. – Ilarraza Sánchez, A. – Alegría Loinaz, I. – Agirre Bengoa, E. Tectogrammar-based machine translation for English-Spanish and English-Basque. *Procesamiento del Lenguaje Natural*. 2016, 56, p. 73–80.

Bahdanau, D. – Cho, K. – Bengio, Y. Neural Machine Translation by Jointly Learning to Align and Translate. In *International Conference on Learning Representations*, San Diego, CA, USA, 2015. arXiv:1409.0473.

Ballesteros, M. – Mille, S. – Wanner, L. Classifiers for data-driven deep sentence generation. In *Proceedings of the 8th International Natural Language Generation Conference*, p. 108–112, Philadelphia, 2014.

Banarescu, L. – Bonial, C. – Cai, S. – Georgescu, M. – Griffitt, K. – Hermjakob, U. – Knight, K. – Koehn, P. – Palmer, M. – Schneider, N. Abstract meaning representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop & Interoperability with Discourse*, p. 178–186, Sofia, 2013.

Bangalore, S. – Rambow, O. Exploiting a probabilistic hierarchical model for generation. In *Proceedings of the 18th conference on Computational linguistics-Volume 1*, p. 42–48, Saarbrücken, Germany, 2000.

Bangalore, S. – Rambow, O. – Whittaker, S. Evaluation metrics for generation. In *Proceedings of the first international conference on Natural language generation-Volume 14*, p. 1–8, Mitzpe Ramon, Israel, 2000.

Bateman, J. A. Enabling technology for multilingual natural language generation: the KPML development environment. *Natural Language Engineering*. 1997, 3, 1, p. 15–55.

Bejček, E. – Panevová, J. – Popelka, J. – Straňák, P. – Ševčíková, M. – Štěpánek, J. – Žabokrtský, Z. Prague Dependency Treebank 2.5 – a revisited version of PDT 2.0. In *Proceedings of the 24th International Conference on Computational Linguistics (Coling 2012)*, p. 231–246, 2012.

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

Belz, A. Statistical generation: Three methods compared and evaluated. In *Proceedings of the 10th European Workshop on Natural Language Generation (ENLG'05)*, p. 15–23, Helsinki, Finland, 2005.

Belz, A. – Kow, E. Assessing the Trade-Off between System Building Cost and Output Quality in Data-to-Text Generation. In Krahmer, E. – Theune, M. (Ed.) *Empirical Methods in Natural Language Generation*, no. 5790 in Lecture Notes in Computer Science. Berlin / Heidelberg, Germany: Springer, 2010a. p. 180–200.

Belz, A. – White, M. – Espinosa, D. – Kow, E. – Hogan, D. – Stent, A. The first surface realisation shared task: Overview and evaluation results. In

*Proceedings of the 13th European Workshop on Natural Language Generation*, p. 217–226, Sofia, Bulgaria, 2011.

Belz, A. – Bohnet, B. – Mille, S. – Wanner, L. – White, M. The Surface Realisation Task: Recent Developments and Future Plans. In *INLG 2012 Proceedings of the 7th International Natural Language Generation Conference*, p. 136–140, Utica, IL, USA, 2012.

Belz, A. – Kow, E. The GREC Challenges 2010: Overview and Evaluation Results. In *Proceedings of the 6th International Natural Language Generation Conference*, p. 219–229, Trim, Ireland, 2010b.

Bengio, S. – Vinyals, O. – Jaitly, N. – Shazeer, N. Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks. In *Advances in Neural Information Processing Systems 28*. Montreal, Canada: Neural Information Processing Systems Foundation, 2015. p. 1171–1179. arXiv: 1506.03099.

Bengio, Y. – Ducharme, R. – Vincent, P. – Jauvin, C. A Neural Probabilistic Language Model. *Journal of Machine Learning Research*. 2003, 3, p. 1137–1155.

Berg, M. M. – Isard, A. – Moore, J. D. An OpenCCG-Based Approach to Question Generation from Concepts. In *Natural Language Processing and Information Systems*, no. 7934 in Lecture Notes in Computer Science. Berlin / Heidelberg, Germany: Springer, June 2013. p. 38–52.

Bilmes, J. A. – Kirchhoff, K. Factored Language Models and Generalized Parallel Backoff. In *Companion volume of the Proceedings of HLT-NAACL 2003 – Short Papers*, p. 4–6, Edmonton, Canada, 2003.

Bishop, C. M. *Pattern recognition and machine learning*. Springer, 2006.

Bohnet, B. – Wanner, L. – Mille, S. – Burga, A. Broad coverage multilingual deep sentence generation with a stochastic multi-level realizer. In *Proceedings of the 23rd International Conference on Computational Linguistics*, p. 98–106, Beijing, China, 2010.

Bohnet, B. – Mille, S. – Favre, B. – Wanner, L. <StuMaBa>: from deep representation to surface. In *Proceedings of the 13th European Workshop on Natural Language Generation*, p. 232–235, Nancy, France, 2011a.

Bohnet, B. – Mille, S. – Wanner, L. Statistical language generation from semantic structures. In *Proceedings of International Conference on Dependency Linguistics*, Barcelona, Spain, 2011b.

Bojar, O. – Žabokrtský, Z. – Dušek, O. – Galuščáková, P. – Majliš, M. – Mareček, D. – Maršík, J. – Novák, M. – Popel, M. – Tamchyna, A. The Joy of Parallelism with CzEng 1.0. In *LREC*, p. 3921–3928, Istanbul, 2012.

Bojar, O. – Buck, C. – Federmann, C. – Haddow, B. – Koehn, P. – Leveling, J. – Monz, C. – Pecina, P. – Post, M. – Saint-Amand, H. – Soricut, R. – Specia, L. – Tamchyna, A. Findings of the 2014 Workshop on Statistical Machine Translation. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, p. 12–58, Baltimore, Maryland, USA, 2014.

Bojar, O. – Chatterjee, R. – Federmann, C. – Haddow, B. – Huck, M. – Hokamp, C. – Koehn, P. – Logacheva, V. – Monz, C. – Negri, M. – Post, M. – Scarton, C. – Specia, L. – Turchi, M. Findings of the 2015 Workshop on Statistical Machine Translation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, p. 1–46, Lisbon, Portugal, 2015.

Bojar, O. – Dušek, O. – Kocmi, T. – Libovický, J. – Novák, M. – Popel, M. – Sudarikov, R. – Variš, D. CzEng 1.6: Enlarged Czech-English Parallel Corpus with Processing Tools Dockered. In *Proceedings of the 19th International Conference on Text, Speech and Dialogue*, Lecture Notes in Artificial Intelligence, Brno, Czech Republic, 2016a.

Bojar, O. – Graham, Y. – Kamran, A. – Stanojević, M. Results of the WMT16 Metrics Shared Task. In *Proceedings of the First Conference on Machine Translation*, p. 199–231, Berlin, Germany, 2016b.

Bojar, O. – Chatterjee, R. – Federmann, C. – Graham, Y. – Haddow, B. – Huck, M. – Yepes, A. J. – Koehn, P. – Logacheva, V. – Monz, C. – others. Findings of the 2016 conference on machine translation (WMT16). In *Proceedings of the First Conference on Machine Translation (WMT), Volume 2: Shared Task Papers*, p. 131–198, Berlin, Germany, 2016c.

Bollmann, M. Adapting SimpleNLG to German. In *Proceedings of the 13th European Workshop on Natural Language Generation*, p. 133–138, 2011.

Brennan, S. E. – Schuhmann, K. S. – Batres, K. M. Entrainment on the move and in the lab: The Walking Around Corpus. In *Proceedings of the 35th Annual Conference of the Cognitive Science Society*, p. 1934–1939, Austin, TX, USA, 2013.

Bresnan, J. *Lexical-Functional Syntax*. Blackwell, 2001.

Brockmann, C. – Isard, A. – Oberlander, J. – White, M. Modelling alignment for affective dialogue. In *Workshop on Adapting the Interaction Style to Affective*

*Factors at the 10th International Conference on User Modeling (UM-05)*, Edinburg, Scotland, UK, 2005.

Buchholz, S. – Marsi, E. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, p. 149–164, New York City, NY, USA, 2006.

Buschmeier, H. – Bergmann, K. – Kopp, S. An alignment-capable microplanner for natural language generation. In *Proceedings of the 12th European Workshop on Natural Language Generation*, p. 82–89, Athens, Greece, 2009.

Buschmeier, H. – Bergmann, K. – Kopp, S. Modelling and Evaluation of Lexical and Syntactic Alignment with a Priming-Based Microplanner. In Krahmer, E. – Theune, M. (Ed.) *Empirical Methods in Natural Language Generation*, no. 5790 in Lecture Notes in Computer Science. Berlin / Heidelberg, Germany: Springer, 2010. p. 85–104.

Cahill, A. – Genabith, J. Robust PCFG-based Generation Using Automatically Acquired LFG Approximations. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics*, p. 1033–1040, Sydney, Australia, 2006.

Callison-Burch, C. – Osborne, M. – Koehn, P. Re-evaluating the role of BLEU in machine translation research. In *11th Conference of the European Chapter of the Association for Computational Linguistics*, p. 249–256, Trento, Italy, 2006.

Callison-Burch, C. – Fordyce, C. – Koehn, P. – Monz, C. – Schroeder, J. (Meta-) evaluation of machine translation. In *Proceedings of the Second Workshop on Statistical Machine Translation*, p. 136–158, Prague, Czech Republic, 2007.

Carenini, G. – Moore, J. D. Generating and evaluating evaluative arguments. *Artificial Intelligence*. 2006, 170, 11, p. 925–952.

Chen, B. – Cherry, C. A systematic comparison of smoothing techniques for sentence-level BLEU. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, p. 362–367, Baltimore, MD, USA, 2014.

Chen, D. L. – Kim, J. – Mooney, R. J. Training a multilingual sportscaster: Using perceptual context to learn language. *Journal of Artificial Intelligence Research*. 2010, 37, p. 397–435.

Cho, K. – Merrienboer, B. – Gulcehre, C. – Bahdanau, D. – Bougares, F. – Schwenk, H. – Bengio, Y. Learning Phrase Representations using RNN

Encoder-Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, p. 1724–1734, Doha, Qatar, 2014. arXiv:1406.1078.

Cho, K. Noisy Parallel Approximate Decoding for Conditional Recurrent Language Model. *arXiv:1605.03835 [cs, stat]*. May 2016.

Chrupała, G. – Dinu, G. – Van Genabith, J. Learning morphology with Morfette. In *Proceedings of LREC*, p. 2362–2367, Marrakech, Morocco, 2008.

Collins, M. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, p. 1–8, Pennsylvania, PA, USA, 2002.

Collins, M. – Duffy, N. New Ranking Algorithms for Parsing and Tagging: Kernels over Discrete Structures, and the Voted Perceptron. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, p. 263–270, Pennsylvania, PA, USA, 2002.

Cotterell, R. – Kirov, C. – Sylak-Glassman, J. – Yarowsky, D. – Eisner, J. – Hulden, M. The SIGMORPHON 2016 shared task—morphological reinflection. In *Proceedings of the 14th Annual SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, p. 10–22, Berlin, Germany, 2016.

Coughlin, D. Correlating automated and human assessments of machine translation quality. In *Proceedings of MT summit IX*, p. 63–70, New Orleans, LA, USA, 2003.

Cristianini, N. – Shawe-Taylor, J. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000.

Dahl, D. A. – Bates, M. – Brown, M. – Fisher, W. – Hunicke-Smith, K. – Pallett, D. – Rudnicky, E. – Shriberg, E. Expanding the scope of the ATIS task: the ATIS-3 corpus. In *Proceedings of the ARPA Human Language Technology Workshop '92*, p. 43–48, Plainsboro, NJ, USA, 1994.

Dale, R. – Scott, D. – Di Eugenio, B. Introduction to the Special Issue on Natural Language Generation. *Computational Linguistics*. September 1998, 24, 3, p. 346–353.

Danescu-Niculescu-Mizil, C. – Lee, L. Chameleons in Imagined Conversations: A New Approach to Understanding Coordination of Linguistic Style in Dialogs. In *Proceedings of the 2nd Workshop on Cognitive Modeling and Computational Linguistics*, p. 76–87, Portland, OR, USA, 2011.

Dannélls, D. On generating coherent multilingual descriptions of museum objects from Semantic Web ontologies. In *INLG 2012 Proceedings of the 7th International Natural Language Generation Conference*, p. 76–84, Utica, IL, USA, 2012.

Kok, D. *Reversible stochastic attribute-value grammars*. PhD thesis, Rijksuniversiteit Groningen, 2013.

Del Gaudio, R. – Burchardt, A. – Aranberri, N. – Branco, A. – Popel, M. Report on the Embedding and Evaluation of the Second MT Pilot. Technical Report Deliverable D3.10, QTLeap, EC FP7 Project no. 610516, 2015.

Dethlefs, N. – Hastie, H. – Cuayáhuitl, H. – Lemon, O. Conditional Random Fields for Responsive Surface Realisation using Global Features. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, p. 1254–1263, Sofia, Bulgaria, 2013.

Devlin, J. – Zbib, R. – Huang, Z. – Lamar, T. – Schwartz, R. – Makhoul, J. Fast and robust neural network joint models for statistical machine translation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, p. 1370–1380, Baltimore, MD, USA, 2014.

Doddington, G. Automatic Evaluation of Machine Translation Quality Using N-gram Co-occurrence Statistics. In *Proceedings of the Second International Conference on Human Language Technology Research*, p. 138–145, San Francisco, CA, USA, 2002.

Dreyer, M. – Eisner, J. Discovering Morphological Paradigms from Plain Text Using a Dirichlet Process Mixture Model. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, p. 616–627, Edinburgh, Scotland, UK, 2011.

Dreyer, M. – Eisner, J. Graphical models over multiple strings. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1*, p. 101–110, Singapore, 2009.

Dreyer, M. – Smith, J. R. – Eisner, J. Latent-variable Modeling of String Transductions with Finite-state Methods. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, p. 1080–1089, Waikiki, HI, USA, 2008.

Dušek, O. – Jurčíček, F. Robust multilingual statistical morphological generation models. In *Proceedings of the ACL Student Research Workshop*, p. 158–164, Sofia, 2013.

Dušek, O. – Jurčíček, F. Training a Natural Language Generator From Unaligned Data. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, p. 451–461, Beijing, China, 2015.

Dušek, O. – Žabokrtský, Z. – Popel, M. – Majliš, M. – Novák, M. – Mareček, D. Formemes in English-Czech deep syntactic MT. In *Proceedings of the Seventh Workshop on Statistical Machine Translation*, p. 267–274, Montreal, Canada, 2012.

Dušek, O. – Gomes, L. – Novák, M. – Popel, M. – Rosa, R. New Language Pairs in TectoMT. In *Proceedings of the 10th Workshop on Machine Translation*, p. 98–104, Lisbon, Portugal, 2015.

Dušek, O. – Jurčíček, F. A Context-aware Natural Language Generation Dataset for Dialogue Systems. In *Proceedings of RE-WOCHAT: Workshop on Collecting and Generating Resources for Chatbots and Conversational Agents – Development and Evaluation*, p. 6–9, Portorož, Slovenia, 2016a.

Dušek, O. – Jurčíček, F. Sequence-to-Sequence Generation for Spoken Dialogue via Deep Syntax Trees and Strings. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, Berlin, Germany, 2016b. arXiv:1606.05491.

Dušek, O. – Jurčíček, F. A Context-aware Natural Language Generator for Dialogue Systems. In *Proceedings of the 17th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, p. 185–190, Los Angeles, CA, USA, 2016c.

Dušek, O. – Plátek, O. – Žilka, L. – Jurčíček, F. Alex: Bootstrapping a Spoken Dialogue System for a New Domain by Real Users. In *Proceedings of the 15th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, p. 79–83, Philadelphia, PA, USA, 2014.

Durrett, G. – DeNero, J. Supervised Learning of Complete Morphological Paradigms. In *Proceedings of NAACL-HLT 2013*, p. 1185–1195, Atlanta, GA, USA, 2013.

Dyer, C. – Weese, J. – Setiawan, H. – Lopez, A. – Ture, F. – Eidelman, V. – Ganitkevitch, J. – Blunsom, P. – Resnik, P. cdec: A decoder, alignment, and learning framework for finite-state and context-free translation models. In *Proceedings of the ACL 2010 System Demonstrations*, p. 7–12, Uppsala, Sweden, 2010.

Eckert, W. – Levin, E. – Pieraccini, R. User modeling for spoken dialogue system evaluation. In *IEEE Workshop on Automatic Speech Recognition and Understanding*, p. 80–87, Santa Barbara, CA, USA, 1997. IEEE.

El Kholy, A. – Habash, N. Rich Morphology Generation Using Statistical Machine Translation. In *INLG 2012 Proceedings of the 7th International Natural Language Generation Conference*, p. 90–94, Utica, IL, USA, 2012.

Elhadad, M. – Robin, J. An overview of SURGE: A reusable comprehensive syntactic realization component. Technical report, Dept. of Computer Science, Ben Gurion University, Beersheba, Israel, 1996.

Eric, M. – Manning, C. D. A Copy-Augmented Sequence-to-Sequence Architecture Gives Good Performance on Task-Oriented Dialogue. *arXiv:1701.04024 [cs]*. January 2017.

Fan, R. E. – Chang, K. W. – Hsieh, C. J. – Wang, X. R. – Lin, C. J. LIBLINEAR: A library for large linear classification. *The Journal of Machine Learning Research*. 2008, 9, p. 1871–1874.

Faruqui, M. – Tsvetkov, Y. – Neubig, G. – Dyer, C. Morphological Inflection Generation Using Character Sequence-to-Sequence Learning. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, p. 634–643, San Diego, CA, USA, 2016. arXiv: 1512.06110.

Federmann, C. Appraise: an open-source toolkit for manual evaluation of MT output. *The Prague Bulletin of Mathematical Linguistics*. 2012, 98, p. 25–35.

Flanigan, J. – Dyer, C. – Smith, N. A. – Carbonell, J. Generation from Abstract Meaning Representation using Tree Transducers. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational*

*Linguistics: Human Language Technologies*, p. 731–739, San Diego, CA, USA, 2016.

Fraser, A. Experiments in Morphosyntactic Processing for Translating to and from German. In *Proceedings of the Fourth Workshop on Statistical Machine Translation*, p. 115–119, Athens, Greece, 2009.

Friedberg, H. – Litman, D. – Paletz, S. B. Lexical entrainment and success in student engineering groups. In *IEEE Spoken Language Technology Workshop*, p. 404–409, Miami, FL, USA, 2012.

Gali, K. – Venkatapathy, S. Sentence Realisation from Bag of Words with dependency constraints. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Student Research Workshop and Doctoral Consortium*, p. 19–24, Boulder, CO, USA, 2009.

Galley, M. – Fosler-Lussier, E. – Potamianos, A. Hybrid natural language generation for spoken dialogue systems. In *Proceedings of the Seventh European Conference on Speech Communication and Technology*, p. 1735–1738, Aalborg, Denmark, 2001.

Galley, M. – Brockett, C. – Sordoni, A. – Ji, Y. – Auli, M. – Quirk, C. – Mitchell, M. – Gao, J. – Dolan, B. deltaBLEU: A discriminative metric for generation tasks with intrinsically diverse targets. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, p. 445–450, Beijing, China, 2015. arXiv:1506.06863.

Gatt, A. – Reiter, E. SimpleNLG: A realisation engine for practical applications. In *Proceedings of the 12th European Workshop on Natural Language Generation*, p. 90–93, Athens, Greece, 2009.

Georgila, K. – Wolters, M. – Moore, J. D. – Logie, R. H. The MATCH corpus: a corpus of older and younger users' interactions with spoken dialogue systems. *Language Resources and Evaluation*. March 2010, 44, 3, p. 221–261.

Ghazvininejad, M. – Brockett, C. – Chang, M.-W. – Dolan, B. – Gao, J. – Yih, W.-T. – Galley, M. A Knowledge-Grounded Neural Conversation Model. *arXiv:1702.01932 [cs]*. February 2017.

Gkatzia, D. – Mahamood, S. A Snapshot of NLG Evaluation Practices 2005 - 2014. In *Proceedings of the 15th European Workshop on Natural Language Generation (ENLG)*, p. 57–60, Brighton, England, UK, 2015.

Goodfellow, I. – Bengio, Y. – Courville, A. *Deep Learning*. MIT Press, 2016. Available at: `http://www.deeplearningbook.org`.

Graham, Y. – Baldwin, T. Testing for Significance of Increased Correlation with Human Judgment. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, p. 172–176, Doha, Qatar, 2014.

Graves, A. Generating Sequences With Recurrent Neural Networks. *arXiv:1308.0850 [cs]*. August 2013.

Gu, J. – Lu, Z. – Li, H. – Li, V. O. K. Incorporating Copying Mechanism in Sequence-to-Sequence Learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, p. 1631–1640, Berlin, Germany, March 2016. arXiv: 1603.06393.

Guo, Y. – Hogan, D. – Van Genabith, J. DCU* at generation challenges 2011 surface realisation track. In *Proceedings of the 13th European workshop on natural language generation*, p. 227–229, Nancy, France, 2011.

Hajič, J. *Disambiguation of rich inflection: computational morphology of Czech*. Karolinum, 2004.

Hajič, J. – Panevová, J. – Hajičová, E. – Sgall, P. – Pajas, P. – Štěpánek, J. – Havelka, J. – Mikulová, M. – Žabokrtský, Z. – Razímová, M. *Prague Dependency Treebank 2.0*. Linguistic Data Consortium, 2006. CD-ROM, LDC Catalog No.: LDC2006T01.

Hajič, J. – Ciaramita, M. – Johansson, R. – Kawahara, D. – Martí, M. A. – Màrquez, L. – Meyers, A. – Nivre, J. – Padó, S. – Štěpánek, J. – others. The CoNLL-2009 shared task: Syntactic and semantic dependencies in multiple languages. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task*, p. 1–18, Boulder, CO, USA, 2009.

Hajič, J. et al. Announcing Prague Czech-English Dependency Treebank 2.0. In *Proceedings of LREC*, p. 3153–3160, Istanbul, Turkey, 2012.

Hart, P. E. – Nilsson, N. J. – Raphael, B. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*. 1968, 4, 2, p. 100–107.

Hastie, H. – Belz, A. A comparative evaluation methodology for NLG in interactive systems. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation*, p. 4004–4011, Reykjavík, Iceland, 2014.

Heafield, K. KenLM: Faster and smaller language model queries. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, p. 187–197, Edinburgh, Scotland, UK, 2011.

Henderson, M. – Thomson, B. – Williams, J. D. The Second Dialog State Tracking Challenge. In *Proceedings of the 15th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*, p. 263–272, Philadelphia, PA, USA, 2014.

Herbrich, R. – Minka, T. – Graepel, T. TrueSkill™: a Bayesian skill rating system. In *Proceedings of the 19th International Conference on Neural Information Processing Systems*, p. 569–576, Vancouver, Canada, 2006. MIT Press.

Hochreiter, S. – Schmidhuber, J. Long short-term memory. *Neural computation*. 1997, 9, 8, p. 1735–1780.

Howcroft, D. M. – Nakatsu, C. – White, M. Enhancing the Expression of Contrast in the SPaRKy Restaurant Corpus. In *Proceedings of the 14th European Workshop on Natural Language Generation*, p. 30–39, Sofia, Bulgaria, 2013.

Hu, Z. – Halberg, G. – Jimenez, C. – Walker, M. Entrainment in pedestrian direction giving: How many kinds of entrainment. In *Proceedings of the IWSDS'2014 Workshop on Spoken Dialog Systems*, p. 90–101, Napa, CA, USA, 2014.

Isard, A. – Brockmann, C. – Oberlander, J. Individuality and alignment in generated dialogues. In *Proceedings of the Fourth International Natural Language Generation Conference*, p. 25–32, Sydney, Australia, 2006.

Jurčíček, F. – Zahradil, J. – Jelínek, L. A human-human train timetable dialogue corpus. In *Proceedings of EUROSPEECH*, p. 1525–1528, Lisbon, Portugal, 2005.

Jurčíček, F. – Keizer, S. – Gašić, M. – Mairesse, F. – Thomson, B. – Yu, K. – Young, S. Real user evaluation of spoken dialogue systems using Amazon Mechanical Turk. In *Proceedings of Interspeech*, p. 3068–3071, Florence, Italy, 2011.

Jurčíček, F. – Dušek, O. – Plátek, O. – Žilka, L. Alex: A Statistical Dialogue Systems Framework. In Sojka, P. – Horák, A. – Kopeček, I. – Pala, K. (Ed.) *Text, Speech and Dialogue: 17th International Conference, TSD*, Lecture Notes in Artificial Intelligence, p. 587–594, Brno, Czech Republic, 2014.

Kann, K. – Schütze, H. Single-Model Encoder-Decoder with Explicit Morphological Representation for Reinflection. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, p. 555–560, Berlin, Germany, 2016. arXiv: 1606.00589.

Karlgren, J. *The Interaction of Discourse Modality and User Expectations in Human-Computer Dialog*. Licentiate Thesis, Stockholm University, 1992.

Kay, M. Chart Generation. In *Proceedings of the 34th Annual Meeting on Association for Computational Linguistics*, p. 200–204, Santa Cruz, CA, USA, 1996.

Kübler, S. – McDonald, R. – Nivre, J. *Dependency Parsing*. No. 2 in Synthesis Lectures on Human Language Technologies. Morgan & Claypool, 2009.

Kiddon, C. – Zettlemoyer, L. L. – Choi, Y. Globally Coherent Text Generation with Neural Checklist Models. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, p. 329–339, Austin, TX, USA, 2016.

Kim, J. – Mooney, R. J. Generative Alignment and Semantic Parsing for Learning from Ambiguous Supervision. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, p. 543–551, Stroudsburg, PA, USA, 2010.

Kim, Y. – Jernite, Y. – Sontag, D. – Rush, A. M. Character-Aware Neural Language Models. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)*, p. 2741–2749, Phoenix, AZ, USA, 2015. arXiv: 1508.06615.

Kingma, D. – Ba, J. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*, Montréal, Canada, 2015. arXiv:1412.6980.

Kneser, R. – Ney, H. Improved backing-off for n-gram language modeling. In *International Conference on Acoustics, Speech, and Signal Processing*, 1, p. 181–184, Detroit, MI, USA, 1995.

Koehn, P. Statistical significance tests for machine translation evaluation. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, p. 388–395, Barcelona, Spain, 2004.

Koehn, P. – Och, F. J. – Marcu, D. Statistical phrase-based translation. In *Proceedings of NAACL-HLT - Volume 1*, p. 48–54, Edmonton, Canada, 2003.

Koehn, P. *Statistical machine translation*. Cambridge University Press, 2010.

Koehn, P. – Hoang, H. – Birch, A. – Callison-Burch, C. – Federico, M. – Bertoldi, N. – Cowan, B. – Shen, W. – Moran, C. – Zens, R. – Dyer, C. – Bojar, O. – Constantin, A. – Herbst, E. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*, p. 187–193, Prague, Czech Republic, 2007.

Konstas, I. – Lapata, M. A Global Model for Concept-to-Text Generation. *Journal of Artificial Intelligence Research*. 2013, 48, p. 305–346.

Korta, K. – Perry, J. Pragmatics. In Zalta, E. N. (Ed.) *The Stanford Encyclopedia of Philosophy*. Stanford, CA, USA: Metaphysics Research Lab, Stanford University, winter 2015 edition, 2015.

Lafferty, J. – McCallum, A. – Pereira, F. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the eighteenth international conference on machine learning, ICML*, 1, p. 282–289, 2001.

Lampouras, G. – Vlachos, A. Imitation learning for language generation from unaligned data. In *The 26th International Conference on Computational Linguistics*, p. 1101–1112, Osaka, Japan, 2016.

Langkilde, I. Forest-based statistical sentence generation. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*, p. 170–177, Seattle, WA, USA, 2000.

Langkilde, I. – Knight, K. Generation that exploits corpus-based statistical knowledge. In *Proceedings of the 36th Annual Meeting of the ACL and 17th International Conference on Computational Linguistics-Volume 1*, p. 704–710, Montréal, Canada, 1998.

Langkilde-Geary, I. An empirical verification of coverage and correctness for a general-purpose sentence generator. In *Proceedings of the 12th International Natural Language Generation Workshop*, p. 17–24, New York City, NY, USA, 2002.

Lavie, A. – Agarwal, A. Meteor: An Automatic Metric for MT Evaluation with High Levels of Correlation with Human Judgments. In *Proceedings of the Second Workshop on Statistical Machine Translation*, p. 228–231, Prague, Czech Republic, 2007.

Lavoie, B. – Rambow, O. A fast and portable realizer for text generation systems. In *Proceedings of the Fifth Conference on Applied Natural Language Processing*, p. 265–268, Washington D.C., USA, 1997.

Lebret, R. – Grangier, D. – Auli, M. Neural Text Generation from Structured Data with Application to the Biography Domain. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, p. 1203–1213, Austin, TX, USA, 2016. arXiv: 1603.07771.

Lemon, O. Adaptive natural language generation in dialogue using Reinforcement Learning. In *Proceedings of SEMdial*, London, England, UK, 2008.

Lemon, O. – Janarthanam, S. – Rieser, V. Statistical Approaches to Adaptive Natural Language Generation. In Lemon, O. – Pietquin, O. (Ed.) *Data-Driven Methods for Adaptive Spoken Dialogue Systems*. New York, NY, USA: Springer, 2012. p. 103–130.

Levenshtein, V. I. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*. 1966, 10, 8, p. 707.

Levitan, R. *Acoustic-Prosodic Entrainment in Human-Human and Human-Computer Dialogue*. PhD thesis, Columbia University, 2014.

Li, J. – Galley, M. – Brockett, C. – Gao, J. – Dolan, B. A Diversity-Promoting Objective Function for Neural Conversation Models. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, p. 110–119, San Diego, CA, USA, 2016a. arXiv: 1510.03055.

Li, J. – Galley, M. – Brockett, C. – Gao, J. – Dolan, B. A Persona-Based Neural Conversation Model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, Berlin, Germany, 2016b. arXiv: 1603.06155.

Li, J. – Monroe, W. – Ritter, A. – Jurafsky, D. Deep Reinforcement Learning for Dialogue Generation. *arXiv:1606.01541 [cs]*. June 2016c.

Liang, P. – Jordan, M. I. – Klein, D. Learning semantic correspondences with less supervision. 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: Volume 1-Volume 1*, p. 91–99, Singapore, 2009.

Lin, C.-Y. ROUGE: A package for automatic evaluation of summaries. In *Text summarization branches out: Proceedings of the ACL-04 workshop*, 8, p. 74–81. Barcelona, Spain, 2004.

Lommel, A. R. – Burchardt, A. – Uszkoreit, H. Multidimensional Quality Metrics: A Flexible System for Assessing Translation Quality. In *Proceedings of ASLIB: Translating and the Computer*, 35, London, UK, 2013.

Lopes, J. – Eskenazi, M. – Trancoso, I. Automated two-way entrainment to improve spoken dialog system performance. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, p. 8372–8376, 2013.

Lopes, J. – Eskenazi, M. – Trancoso, I. From rule-based to data-driven lexical entrainment models in spoken dialog systems. *Computer Speech & Language*. 2015, 31, 1, p. 87–112.

Lowe, R. – Pow, N. – Serban, I. – Pineau, J. The Ubuntu Dialogue Corpus: A Large Dataset for Research in Unstructured Multi-Turn Dialogue Systems. In *Proceedings of the 16th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, p. 285–294, Prague, Czech Republic, 2015.

Lu, W. – Ng, H. T. – Lee, W. S. Natural language generation with tree conditional random fields. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1-Volume 1*, p. 400–409, Singapore, 2009.

Luan, Y. – Ji, Y. – Ostendorf, M. LSTM based Conversation Models. *arXiv:1603.09457 [cs]*. March 2016.

Luong, M.-T. – Manning, C. D. Achieving Open Vocabulary Neural Machine Translation with Hybrid Word-Character Models. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, p. 1054–1063, Berlin, Germany, 2016. arXiv: 1604.00788.

Luong, M.-T. – Pham, H. – Manning, C. D. Effective Approaches to Attention-based Neural Machine Translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, p. 1412–1421, Lisbon, Portugal, 2015. arXiv: 1508.04025.

Mairesse, F. – Walker, M. Trainable generation of big-five personality styles through data-driven parameter estimation. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies*, p. 165–173, Columbus, OH, USA, 2008.

Mairesse, F. – Walker, M. PERSONAGE: Personality generation for dialogue. In *Proceedings of the 45th Annual Meeting of the Association For Computational Linguistics*, p. 496–503, Prague, Czech Republic, 2007.

Mairesse, F. – Walker, M. Controlling User Perceptions of Linguistic Style: Trainable Generation of Personality Traits. *Computational Linguistics*. 2011, 37, 3, p. 455–488.

Mairesse, F. – Gašić, M. – Jurčíček, F. – Keizer, S. – Thomson, B. – Yu, K. – Young, S. Phrase-based statistical language generation using graphical models and active learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, p. 1552–1561, Uppsala, Sweden, 2010.

Manishina, E. – Jabaian, B. – Huet, S. – Lefevre, F. Automatic Corpus Extension for Data-driven Natural Language Generation. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, p. 3624–3631, Paris, France, 2016. European Language Resources Association (ELRA).

Manning, C. D. – Schütze, H. *Foundations of statistical natural language processing*. MIT Press, 2000.

Marcus, M. P. – Marcinkiewicz, M. A. – Santorini, B. Building a large annotated corpus of English: The Penn Treebank. *Computational linguistics*. 1993, 19, 2, p. 330.

Mareček, D. – Popel, M. – Žabokrtský, Z. Maximum entropy translation model in dependency-based MT framework. In *Proceedings of the Joint Fifth Workshop on Statistical Machine Translation and MetricsMATR*, p. 201–206, Uppsala, Sweden, 2010.

Mazzei, A. – Battaglino, C. – Bosco, C. SimpleNLG-IT: adapting SimpleNLG to Italian. In *The 9th International Natural Language Generation conference*, p. 184–192, Edinburgh, Scotland, UK, 2016.

McDonald, R. – Pereira, F. – Ribarov, K. – Hajič, J. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, p. 523–530, Vancouver, Canada, 2005.

McDonald, R. – Hall, K. – Mann, G. 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*, p. 456–464, 2010.

Mei, H. – Bansal, M. – Walter, M. R. What to talk about and how? Selective Generation using LSTMs with Coarse-to-Fine Alignment. In *The 15th Annual*

*Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, p. 720–730, San Diego, CA, USA, 2016. arXiv: 1509.00838.

Melčuk, I. A. *Dependency syntax: theory and practice*. SUNY series in linguistics. State University Press of New York, 1988.

Meteer, M. W. *The "generation gap": The problem of expressibility in text planning*. Ph.D. thesis, University of Massachusetts, 1990.

Mikolov, T. – Kombrink, S. – Burget, L. – Černocký, J. – Khudanpur, S. Extensions of recurrent neural network language model. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, p. 5528–5531, Prague, Czech Republic, 2011.

Mikolov, T. – Karafiát, M. – Burget, L. – Černocký, J. – Khudanpur, S. Recurrent neural network based language model. In *Proceedings of Interspeech*, p. 1045–1048, Makuhari, Japan, 2010.

Minnen, G. – Carroll, J. – Pearce, D. Applied morphological processing of English. *Natural Language Engineering*. 2001, 7, 3, p. 207–223.

Mitchell, M. – Bohus, D. – Kamar, E. Crowdsourcing Language Generation Templates for Dialogue Systems. In *Proceedings of the INLG and SIGDIAL 2014 Joint Session*, p. 24–32, Philadelphia, PA, USA, 2014.

Moore, J. – Foster, M. E. – Lemon, O. – White, M. Generating Tailored, Comparative Descriptions in Spoken Dialogue. In *Proceedings of FLAIRS*, Miami Beach, FL, USA, 2004.

Nakanishi, H. – Miyao, Y. – Tsujii, J. Probabilistic Models for Disambiguation of an HPSG-based Chart Generator. In *Proceedings of the Ninth International Workshop on Parsing Technology*, p. 93–102, Vancouver, Canada, 2005.

Nakatsu, C. – White, M. Learning to say it well: reranking realizations by predicted synthesis quality. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the ACL*, p. 1113–1120, Sydney, Australia, 2006.

Naughton, J. *Czech : An essential grammar*. Routledge, 2005.

Nenkova, A. – Gravano, A. – Hirschberg, J. High frequency word entrainment in spoken dialogue. In *Proceedings of the 46th Annual Meeting of the Association*

*for Computational Linguistics on Human Language Technologies: Short Papers*, p. 169–172, Columbus, OH, USA, 2008.

Nenkova, A. – Chae, J. – Louis, A. – Pitler, E. Structural features for predicting the linguistic quality of text. In *Empirical methods in natural language generation*. Cambridge, MA, USA: Springer, 2010. p. 222–241.

Nicolai, G. – Cherry, C. – Kondrak, G. Inflection Generation as Discriminative String Transduction. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, p. 922–931, Denver, CO, USA, 2015.

Nilsson, J. – Riedel, S. – Yuret, D. The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the CoNLL shared task session of EMNLP-CoNLL*, p. 915–932, Prague, Czech Republic, 2007.

Nivre, J. – Marneffe, M.-C. d. – Ginter, F. – Goldberg, Y. – Hajič, J. – Manning, C. D. – McDonald, R. – Petrov, S. – Pyysalo, S. – Silveira, N. – Tsarfaty, R. – Zeman, D. Universal Dependencies v1: A Multilingual Treebank Collection. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, p. 1659–1666, Portorož, Slovenia, 2016.

Novák, V. – Žabokrtský, Z. Feature engineering in maximum spanning tree dependency parser. In *Text, Speech and Dialogue*, p. 92–98, Plzeň, Czech Republic, 2007.

Och, F. J. – Ney, H. A Systematic Comparison of Various Statistical Alignment Models. *Computational Linguistics*. 2003, 29, 1, p. 19–51.

Och, F. J. – Ueffing, N. – Ney, H. An Efficient A* Search Algorithm for Statistical Machine Translation. In *Proceedings of the Workshop on Data-driven Methods in Machine Translation - Volume 14*, p. 1–8, Toulouse, France, 2001.

Oh, A. H. – Rudnicky, A. I. Stochastic language generation for spoken dialogue systems. In *Proceedings of the 2000 ANLP/NAACL Workshop on Conversational systems-Volume 3*, p. 27–32, Seattle, WA, USA, 2000.

Paiva, D. S. – Evans, R. Empirically-based control of natural language generation. In *Proceedings of the 43rd Annual Meeting of ACL*, p. 58–65, Stroudsburg, PA, USA, 2005. ACL.

Papineni, K. – Roukos, S. – Ward, T. – Zhu, W.-J. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting*

*of the Association for Computational Linguistics*, p. 311–318, Pennsylvania, PA, USA, 2002.

Parent, G. – Eskenazi, M. Lexical Entrainment of Real Users in the Let's Go Spoken Dialog System. In *Proceedings of Interspeech*, p. 3018–3021, Makuhari, Japan, 2010.

Pavlick, E. – Post, M. – Irvine, A. – Kachaev, D. – Callison-Burch, C. The language demographics of Amazon Mechanical Turk. *Transactions of the Association for Computational Linguistics*. 2014, 2, p. 79–92.

Pollard, C. – Sag, I. A. *Head-Driven Phrase Structure Grammar*. University of Chicago Press, 1994.

Popel, M. – Žabokrtský, Z. Improving English-Czech Tectogrammatical MT. *The Prague Bulletin of Mathematical Linguistics*. 2009, 92, p. 115–134.

Popel, M. – Žabokrtský, Z. TectoMT: modular NLP framework. In *Proceedings of IceTAL, 7th International Conference on Natural Language Processing*, p. 293–304, Reykjavík, 2010.

Popel, M. *Ways to Improve the Quality of English-Czech Machine Translation*. Master's thesis, Charles University in Prague, 2009.

Popel, M. et al. Report on the Second MT Pilot and Its Evaluation. Technical Report Deliverable D2.8, QTLeap, EC FP7 Project no. 610516, 2015a.

Popel, M. et al. Report on the first MT pilot and its evaluation. Technical Report Deliverable D2.4, QTLeap, EC FP7 Project no. 610516, 2015b.

Ptáček, J. Two Tectogrammatical Realizers Side by Side: Case of English and Czech. In *Fourth International Workshop on Human-Computer Conversation*, Bellagio, Italy, 2008.

Ptáček, J. – Žabokrtský, Z. Dependency-Based Sentence Synthesis Component for Czech. In *Proceedings of 3rd International Conference on Meaning-Text Theory*, 69 / *Wiener Slawistischer Almanach*, p. 407–415. Verlag Otto Sagner, 2007.

Ptáček, J. – Žabokrtský, Z. Synthesis of Czech Sentences from Tectogrammatical Trees. In *Text, Speech and Dialogue*, p. 221–228, Brno, Czech Republic, 2006.

Rajkumar, R. – Espinosa, D. – White, M. The OSU system for surface realization at generation challenges 2011. In *Proceedings of the 13th European Workshop on Natural Language Generation*, p. 236–238, Nancy, France, 2011.

Rambow, O. – Bangalore, S. – Walker, M. Natural language generation in dialog systems. In *Proceedings of the First International Conference on Human Language Technology Research*, p. 1–4, San Diego, CA, USA, 2001.

Ratnaparkhi, A. Trainable methods for surface natural language generation. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*, p. 194–201, Seattle, WA, USA, 2000.

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

Raux, A. – Langner, B. – Bohus, D. – Black, A. W. – Eskenazi, M. Let's go public! taking a spoken dialog system to the real world. In *Proceedings. of Interspeech 2005*, Lisbon, Portugal, 2005.

Raux, A. – Langner, B. – Black, A. W. – Eskenazi, M. LET'S GO: Improving Spoken Dialog Systems for the Elderly and Non-natives. In *Proceedings of the 8th European Conference on Speech Communication and Technology*, p. 753–756, Geneva, Switzerland, 2003. International Speech Communication Association.

Reiter, E. – Dale, R. *Building Natural Language Generation Systems*. Cambridge University Press, 2000.

Reiter, E. – Sripada, S. – Hunter, J. – Yu, J. – Davy, I. Choosing words in computer-generated weather forecasts. *Artificial Intelligence*. 2005, 167, 1-2, p. 137–169.

Reiter, E. – Belz, A. An Investigation into the Validity of Some Metrics for Automatically Evaluating Natural Language Generation Systems. *Computational Linguistics*. 2009, 35, 4, p. 529–558.

Reitter, D. – Keller, F. – Moore, J. D. Computational Modelling of Structural Priming in Dialogue. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, p. 121–124, New York City, NY, USA, 2006.

Resnik, P. – Lin, J. Evaluation of NLP systems. In Clark, A. – Fox, C. – Lappin, S. (Ed.) *The Handbook of Computational Linguistics and Natural Language Processing*, Blackwell Handbooks in Linguistics. Chichester UK / Malden, MA, USA: Wiley-Blackwell, 2010. p. 271–296.

Rieser, V. – Lemon, O. Natural language generation as planning under uncertainty for spoken dialogue systems. In *Empirical methods in natural language generation*, p. 105–120, 2010.

Rieser, V. – Lemon, O. – Liu, X. Optimising information presentation for spoken dialogue systems. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, p. 1009–1018, Uppsala, Sweden, 2010.

Rosa, R. – Dušek, O. – Mareček, D. – Popel, M. Using Parallel Features in Parsing of Machine-Translated Sentences for Correction of Grammatical Errors. In *Proceedings of Sixth Workshop on Syntax, Semantics and Structure in Statistical Translation (SSST-6), ACL*, p. 39–48, Jeju, Korea, 2012.

Rosa, R. – Dušek, O. – Novák, M. – Popel, M. Translation Model Interpolation for Domain Adaptation in TectoMT. In *Proceedings of the 1st Deep Machine Translation Workshop*, p. 89–96, Prague, Czech Republic, 2015. ÚFAL MFF UK.

Rudnicky, A. I. – Thayer, E. H. – Constantinides, P. C. – Tchou, C. – Shern, R. – Lenzo, K. A. – Xu, W. – Oh, A. Creating natural dialogs in the Carnegie Mellon Communicator system. In *Proceedings of the 6th European Conference on Speech Communication and Technology*, p. 1531–1534, Lisbon, Portugal, 1999.

Sakaguchi, K. – Post, M. – Van Durme, B. Efficient elicitation of annotations for human evaluation of machine translation. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, p. 1–11, Baltimore, MD, USA, 2014.

Sennrich, R. – Haddow, B. – Birch, A. Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, p. 1715–1725, Berlin, Germany, 2016. arXiv: 1508.07909.

Serban, I. V. – Sordoni, A. – Bengio, Y. – Courville, A. – Pineau, J. Building End-To-End Dialogue Systems Using Generative Hierarchical Neural Network Models. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, Phoenix, AZ, USA, 2016. arXiv: 1507.04808.

Serban, I. V. – Lowe, R. – Charlin, L. – Pineau, J. A Survey of Available Corpora for Building Data-Driven Dialogue Systems. *arXiv:1512.05742 [cs, stat]*. December 2015.

Sgall, P. – Hajičová, E. – Panevová, J. *The meaning of the sentence in its semantic and pragmatic aspects*. D. Reidel, 1986.

Shapiro, D. – Langley, P. Separating Skills from Preference: Using Learning to Program by Reward. In *Proceedings of the 19th International Conference on Machine Learning (ICML)*, p. 1–8, Sydney, Australia, 2002.

Sharma, S. – He, J. – Suleman, K. – Schulz, H. – Bachman, P. Natural Language Generation in Dialogue using Lexicalized and Delexicalized Data. *arXiv:1606.03632 [cs]*. June 2016.

Spoustová, D. J. – Hajič, J. – Votrubec, J. – Krbec, P. – Květoň, P. The Best of Two Worlds: Cooperation of Statistical and Rule-based Taggers for Czech. In *Proceedings of the Workshop on Balto-Slavonic Natural Language Processing: Information Extraction and Enabling Technologies*, p. 67–74, Prague, Czech Republic, 2007.

Sripada, S. G. – Reiter, E. – Hunter, J. – Yu, J. Exploiting a parallel text-data corpus. In *Proceedings of the Corpus Linguistics 2003 conference*, p. 734–743, Lancaster, England, UK, 2003.

Stanojević, M. – Kamran, A. – Koehn, P. – Bojar, O. Results of the WMT15 Metrics Shared Task. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, p. 256–273, Lisbon, Portugal, 2015.

Steedman, M. *The syntactic process*. MIT Press, 2000.

Stent, A. ATT-0: Submission to generation challenges 2011 surface realization shared task. In *Proceedings of the Generation Challenges Session at the 13th European Workshop on Natural Language Generation*, p. 230–231, Nancy, France, 2011.

Stent, A. – Prasad, R. – Walker, M. Trainable sentence planning for complex information presentation in spoken dialog systems. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, p. 79–86, Barcelona, Spain, 2004.

Stent, A. – Marge, M. – Singhai, M. Evaluating evaluation methods for generation in the presence of variation. In *International Conference on Intelligent Text Processing and Computational Linguistics*, p. 341–351, Mexico City, Mexico, 2005.

Stoyanchev, S. – Stent, A. Lexical and Syntactic Priming and Their Impact in Deployed Spoken Dialog Systems. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for*

*Computational Linguistics, Companion Volume: Short Papers*, p. 189–192, Boulder, CO, USA, 2009.

Straková, J. – Straka, M. – Hajič, J. Open-Source Tools for Morphology, Lemmatization, POS Tagging and Named Entity Recognition. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, p. 13–18, Baltimore, MA, USA, 2014.

Sutskever, I. – Vinyals, O. – Le, Q. V. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, p. 3104–3112, Montréal, Canada, 2014. arXiv:1409.3215.

Sutton, C. – McCallum, A. An introduction to conditional random fields. *Foundations and Trends® in Machine Learning*. 2012, 4, 4, p. 267–373.

Sutton, R. S. – Barto, A. G. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

Sylak-Glassman, J. – Kirov, C. – Yarowsky, D. – Que, R. A Language-Independent Feature Schema for Inflectional Morphology. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, p. 674–680, Beijing, China, 2015.

Toutanova, K. – Suzuki, H. – Ruopp, A. Applying morphology generation models to machine translation. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies*, p. 514–522, Columbus, OH, USA, 2008.

Deemter, K. – Krahmer, E. – Theune, M. Real vs. template-based natural language generation: a false opposition? *Computational Linguistics*. 2005, 31, 1, p. 15–24.

Deemter, K. – Sluis, I. – Gatt, A. Building a Semantically Transparent Corpus for the Generation of Referring Expressions. In *Proceedings of the Fourth International Natural Language Generation Conference*, p. 130–132, Sydney, Australia, 2006.

Van Noord, G. At last parsing is now operational. In *Verbum Ex Machina (TALN vol. 1): Actes de la 13e conference sur le traitement automatique des langues naturelles*, p. 20–42, Leuven, Belgium, 2006.

Vaswani, A. – Zhao, Y. – Fossum, V. – Chiang, D. Decoding with Large-Scale Neural Language Models Improves Translation. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, p. 1387–1392, Seattle, WA, USA, 2013.

Vaudry, P.-L. – Lapalme, G. Adapting SimpleNLG for bilingual English-French realisation. In *Proceedings of the 14th European Workshop on Natural Language Generation*, p. 183–187, Sofia, Bulgaria, 2013.

Vejman, M. *Development of an English public transport information dialogue system*. Master's thesis, Charles University in Prague, 2015.

Viethen, J. – Dale, R. The Use of Spatial Relations in Referring Expression Generation. In *Proceedings of the Fifth International Natural Language Generation Conference*, p. 59–67, Salt Fork, OH, USA, 2008.

Vinyals, O. – Le, Q. A Neural Conversational Model. In *Proceedings of the 31st International Conference on Machine Learning*, Lille, France, June 2015. arXiv:1506.05869.

Vinyals, O. – Kaiser, Ł. – Koo, T. – Petrov, S. – Sutskever, I. – Hinton, G. Grammar as a Foreign Language. In *Advances in Neural Information Processing Systems 28*, p. 2755–2763, Montréal, Canada, 2015a.

Vinyals, O. – Toshev, A. – Bengio, S. – Erhan, D. Show and Tell: A Neural Image Caption Generator. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, p. 3156–3164, Boston, MA, USA, 2015b. arXiv: 1411.4555.

Walker, M. – Whittaker, S. – Stent, A. – Maloor, P. – Moore, J. – Johnston, M. – Vasireddy, G. Generation and evaluation of user tailored responses in multimodal dialogue. *Cognitive Science*. 2004, 28, 5, p. 811–840.

Walker, M. A. – Rambow, O. – Rogati, M. SPoT: a trainable sentence planner. In *Proceedings of 2nd meeting of NAACL*, p. 1–8, Stroudsburg, PA, USA, 2001a. ACL.

Walker, M. – Passonneau, R. DATE: a dialogue act tagging scheme for evaluation of spoken dialogue systems. In *Proceedings of the First International Conference on Human Language Technology Research*, San Diego, CA, USA, 2001.

Walker, M. – Stent, A. – Mairesse, F. – Prasad, R. Individual and Domain Adaptation in Sentence Planning for Dialogue. *Journal of Artificial Intelligence Research*. 2007, 30, p. 413–456.

Walker, M. A. – Passonneau, R. – Boland, J. E. Quantitative and qualitative evaluation of DARPA Communicator spoken dialogue systems. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, p. 515–522, Toulouse, France, 2001b.

Walker, M. A. – Rambow, O. C. – Rogati, M. Training a sentence planner for spoken dialogue using boosting. *Computer Speech & Language*. July 2002, 16, 3–4, p. 409–433.

Wang, Z. – He, W. – Wu, H. – Wu, H. – Li, W. – Wang, H. – Chen, E. Chinese Poetry Generation with Planning based Neural Network. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, p. 1051–1060, Osaka, Japan, 2016. arXiv: 1610.09889.

Wen, T.-H. – Gašić, M. – Mrkšić, N. – Su, P.-H. – Vandyke, D. – Young, S. Semantically Conditioned LSTM-based Natural Language Generation for Spoken Dialogue Systems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, p. 1711–1721, Lisbon, Portugal, 2015a.

Wen, T.-H. – Gasic, M. – Kim, D. – Mrksic, N. – Su, P.-H. – Vandyke, D. – Young, S. Stochastic Language Generation in Dialogue using Recurrent Neural Networks with Convolutional Sentence Reranking. In *Proceedings of the 16th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, p. 275–284, Prague, Czech Republic, 2015b.

Wen, T.-H. – Gasic, M. – Mrksic, N. – Rojas-Barahona, L. M. – Su, P.-H. – Ultes, S. – Vandyke, D. – Young, S. A Network-based End-to-End Trainable Task-oriented Dialogue System. *arXiv:1604.04562 [cs, stat]*. April 2016a.

Wen, T.-H. – Gasic, M. – Mrksic, N. – Rojas-Barahona, L. M. – Su, P.-H. – Ultes, S. – Vandyke, D. – Young, S. Conditional Generation and Snapshot Learning in Neural Dialogue Systems. *arXiv:1606.03352 [cs, stat]*. June 2016b.

Wen, T.-H. – Gasic, M. – Mrksic, N. – Rojas-Barahona, L. M. – Su, P.-H. – Vandyke, D. – Young, S. Multi-domain Neural Network Language Generation for Spoken Dialogue Systems. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, p. 120–129, San Diego, CA, USA, 2016c. arXiv: 1603.01232.

White, M. – Baldridge, J. Adapting chart realization to CCG. In *Proceedings of the 9th European Workshop on Natural Language Generation*, p. 119–126, Budapest, Hungary, 2003.

White, M. – Rajkumar, R. Perceptron reranking for CCG realization. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1-Volume 1*, p. 410–419, Singapore, 2009.

White, M. – Rajkumar, R. – Martin, S. Towards broad coverage surface realization with CCG. In *Proceedings of the Workshop on Using Corpora for NLG: Language Generation and Machine Translation (UCNLG+MT)*, p. 22–30, Copenhagen, Denmark, 2007.

Williams, J. – Raux, A. – Ramachadran, D. – Black, A. The Dialog State Tracking Challenge. In *Proceedings of the 14th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, p. 404–413, Metz, France, 2013.

Williams, J. D. – Asadi, K. – Zweig, G. Hybrid Code Networks: practical and efficient end-to-end dialog control with supervised and reinforcement learning. *arXiv:1702.03274 [cs]*. February 2017.

Wong, Y. W. – Mooney, R. J. Generation by inverting a semantic parser that uses statistical machine translation. In *Proceedings of Human Language Technologies: The Conference of the North American Chapter of the ACL (NAACL-HLT-07)*, p. 172–179, Prague, Czech Republic, 2007.

Xing, C. – Wu, W. – Wu, Y. – Liu, J. – Huang, Y. – Zhou, M. – Ma, W.-Y. Topic Augmented Neural Response Generation with a Joint Attention Mechanism. *arXiv:1606.08340 [cs]*. June 2016.

Yi, X. – Li, R. – Sun, M. Generating Chinese Classical Poems with RNN Encoder-Decoder. *arXiv:1604.01537 [cs]*. April 2016.

Young, S. – Gašić, M. – Keizer, S. – Mairesse, F. – Schatzmann, J. – Thomson, B. – Yu, K. The Hidden Information State model: A practical framework for POMDP-based spoken dialogue management. *Computer Speech & Language*. 2010, 24, 2, p. 150–174.

Young, S. – Gašić, M. – Thomson, B. – Williams, J. POMDP-Based Statistical Spoken Dialog Systems: A Review. *Proceedings of the IEEE*. May 2013, 101, 5, p. 1160–1179.

Young, S. CUED standard dialogue acts. Technical report, Cambridge University Engineering Department, Cambridge, England, UK, 2009.

Žabokrtský, Z. – Ptáček, J. – Pajas, P. TectoMT: highly modular MT system with tectogrammatics used as transfer layer. In *Proceedings of the Third Workshop on Statistical Machine Translation*, p. 167–170, Columbus, OH, USA, 2008.

Zeman, D. – Mareček, D. – Popel, M. – Ramasamy, L. – Štěpánek, J. – Zabokrtský, Z. – Hajič, J. HamleDT: To Parse or Not to Parse? In *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, p. 2735–2741, 2012.

Zeman, D. – Dušek, O. – Mareček, D. – Popel, M. – Ramasamy, L. – Štěpánek, J. – Žabokrtský, Z. – Hajič, J. HamleDT: Harmonized Multi-Language Dependency Treebank. *Language Resources and Evaluation*. 2014, 48, 4, p. 601–637.

Zettlemoyer, L. S. – Collins, M. 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*, p. 678–687, Prague, Czech Republic, 2007.

Zhang, X. – Lapata, M. Chinese poetry generation with recurrent neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, p. 670–680, Doha, Qatar, 2014.

Zhang, Y. – Clark, S. Syntax-based grammaticality improvement using CCG and guided search. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, p. 1147–1157, Edinburgh, Scotland, UK, 2011.

# List of Abbreviations

| | |
|---|---|
| AMR | Abstract Meaning Representation (Banarescu et al., 2013) |
| API | Application programming interface |
| CCG | Combinatory categorial grammar |
| CFG | Context-free grammar |
| CoNLL | Conference on Natural Language Learning |
| CRF | Conditional random field (Lafferty et al., 2001) |
| DA | Dialogue act |
| DAI | Dialogue act item (a triplet of DA type, slot, value) |
| ERR | Slot error rate (Wen et al., 2015a) |
| FGD | Functional Generative Description (Sgall et al., 1986) |
| HPSG | Head-driven phrase structure grammar (Pollard and Sag, 1994) |
| IT | Information technology |
| LM | Language model |
| MR | Meaning representation |
| MST | Maximum Spanning Tree (parser algorithm of McDonald et al., 2005) |
| MT | Machine translation |
| NLG | Natural language generation |
| NLP | Natural language processing |
| NN | Neural network |
| PDT | Prague Dependency Treebank (Hajič et al., 2006; Bejček et al., 2012) |

| | |
|---|---|
| RNN | Recurrent neural network |
| SDS | Spoken dialogue system |
| seq2seq | Sequence-to-sequence |
| SGD | Stochastic gradient descent |
| SVM | Support vector machine (Cristianini and Shawe-Taylor, 2000) |
| SVP | Slot-value pair (in dialogue acts) |
| WMT | Workshop on Statistical Machine Translation |