NATURAL LANGUAGE GENERATION USING AN INFORMATION-SLIM REPRESENTATION

by

Radu Soricut

A Dissertation Presented to the
FACULTY OF THE GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(COMPUTER SCIENCE)

August 2006

Dedication

To my wife, Ioana

Acknowledgments

This thesis would not have been possible without the direct or indirect contributions of many people who provided me with their support.

First and foremost, I wish to thank my adviser, Daniel Marcu, for the guidance he provided me throughout my entire doctoral research. He taught me a lot about what it means to do research, and I will probably do research my entire life the way he taught me. I also want to thank Kevin Knight for all the suggestions and ideas he bounced to me during these years, many of which ended up incorporated in this thesis.

I also extend my thanks to rest of my thesis committee – Eduard Hovy, Paul Rosenbloom, Daniel O'Leary – for all their suggestions toward improving the coverage of my work.

I also want to use this opportunity to thank my former advisers for bringing me where I am today. Gheorghe Stefanescu was my undergraduate adviser at the University of Bucharest, and I owe him for the patience he took to initiate me into doing quality research. Soon afterward, during my first research project at UNU/IIST, Kees Middelburg and Jan Bergstra helped me broaden my horizon, and facilitated my exposure to key elements in the field of formal methods, which proved valuable later in my career. For all they have done for me, many thanks.

An important role during my graduate studies was played by my fellow graduate students in the Natural Language Group at ISI. I collaborated with many of them – Hal Daume III, Abdessamad Echihabi, Alex Fraser, Dragos Stefan Munteanu, Liang Zhou – and I have had many fruitful discussions about and beyond my work.

I thoroughly enjoyed my summer internship at Microsoft Research, where I had the opportunity to diversify my experience and knowledge. I am indebted to Eric Brill for the opportunity to work with him, and for providing me with good feedback and guidance.

Finally, I want to express my deepest gratitude for my family. My wife, Ioana, has been near me for all these years, supported and eased my work relentlessly, and made possible for me to achieve my goal. My two daughters, Andreea Ioana and Alexandra Clara, have given me all the joy I needed, and all the satisfaction I wanted.

Contents

List Of Tables

List Of Figures

Abstract

In this dissertation, I propose a new natural language generation paradigm, based on direct transformation of textual information into well-formed textual output. I support this language generation paradigm with theoretical contributions in the field of formal languages, new algorithms, empirical results, and software implementations. At the core of this work is a novel representation formalism for probability distributions over finite languages. Due to its convenient representation and computational properties, this formalism supports a wide range of language generation needs, from sentence realization to text planning.

Based on this formalism, I describe, implement, and analyze theoretically a family of algorithms that perform language generation using direct transformations of text. These algorithms use stochastic models of language to drive the generation process. I perform extensive empirical evaluations using my implementation of these algorithms. These evaluations show state-of-the-art performance in automatic translation, and significant improvements in state-of-the-art performance in abstractive headline generation and coherent discourse generation.

Chapter 1

Introduction

1.1 Natural Language Generation for Text-to-Text Applications

The long-term research goal that drives the research presented in this dissertation is to make computers achieve human levels of performance when using natural language. More precisely, I am interested in developing theories and algorithms that can lead to computer software capable of producing meaningful sentences and structured, coherent text from bits and pieces of textual information. These capabilities, deployed in applications such as automatic translation, summarization, and question answering, could significantly increase our ability to assimilate and disseminate information.

The focus of this dissertation is on the problem of using natural language to present information, a task called Natural Language Generation (NLG). Over the years, research efforts in NLG have produced a considerable number of generation systems: Penman (Matthiessen & Bateman 1991), FUF (Elhadad 1991), Nitrogen (Knight & Hatzivassiloglou 1995), Fergus (Bangalore & Rambow 2000b), HALogen (Langkilde-Geary 2002), Amalgam (Corston-Oliver *et al.* 2002),

etc. However, when it comes to end-to-end, text-to-text applications – Machine Translation, Summarization, Question Answering – none of these generic systems is usually employed. In Machine Translation (Germann *et al.* 2001; Och & Ney 2002), state-of-the-art translation systems use application-specific models and "decoders", inspired by work in speech recognition (Brown *et al.* 1993), while generation-heavy approaches produce translations of a much lower quality (Hajic *et al.* 2002; Habash 2003). In the Automatic Summarization field, summaries are generated as either extracts of human-produced texts (Hovy & Lin 2000), or using task-specific strategies (Barzilay 2003). In Question Answering, answers that need long formulations are also presented via extracts from human-produced texts (Soricut & Brill 2004). I believe two reasons explain this state of affairs.

First, these generic NLG systems use input representation formalisms with complex syntax and semantics, and rigid built-in assumptions. These languages involve deep, semantic-based subject-verb or verb-object relations (such as ACTOR, AGENT, PATIENT, POSSESSOR, etc., for Penman and FUF), syntactic relations (such as `subject, object, premod`, etc., for HALogen), or lexical dependencies (Fergus, Amalgam). Such input representations cannot be easily produced by state-of-the-art analysis components from arbitrary text inputs in the context of text-to-text applications. For instance, a summarization application that needs to generate a short summary headline from an input text document will have difficulties automatically creating input representations using information that spans multiple sentences, for any of these representation formalisms. Moreover, the granularity level of these representations is built-in

and rigid (usually at word level), which makes it difficult to capture phenomena at different granularity levels (for instance, phrase-level translation correspondences).

Second, most of the recent systems (starting with Nitrogen) have adopted a hybrid approach to generation, which has increased their robustness. These hybrid systems use, in a first phase, symbolic knowledge to (over)generate a large set of candidate realizations, and, in a second phase, statistical knowledge about the target language (such as stochastic language models) to rank the candidate realizations and find the best scoring one. The disadvantage of the hybrid approach – from the perspective of integrating these systems within end-to-end applications – is that the two generation phases cannot be tightly coupled in a true probabilistic model. More precisely, if some candidates proposed in the first phase are to be preferred to some other candidates for reasons that concern solely the input, the hybrid systems mentioned above have no principled mechanism to integrate the weights of these preferences with the weights of the target language preferences. Some of these systems do not address the preference integration problem al all, while those that do address it (Langkilde-Geary 2002) have limited expressive power (e.g., only preferences for lexical choices can be expressed, while input-biased preferences for word reordering cannot).

1.2 Natural Language Generation using WIDL-expressions

In this dissertation, I propose a new language generation paradigm, based on direct transformation of textual information into well-formed textual output. Consider, for instance, a user who wants to be informed, in only one short sentence, of the content of one or more documents, either

as a generic summary or as an answer to some specific question. State-of-the-art text analyzers are capable of identifying accurately significant words in the input text (Zhou & Hovy 2003; Zajic, Dorr, & Schwartz 2004). For example, after processing documents that discuss the latest developments of floods in China, such an analyzer builds a list of relevant terms containing Wuhan, Yangtze, chinese, river, rain, and reaches. The challenge in this generation paradigm is to automatically produce a correct and meaningful sentence starting from such list. A shortcut frequently used by current applications is to extract a sentence from the input documents that fits best the relevant term list (Dorr, Zajic, & Schwartz 2003; Zajic, Dorr, & Schwartz 2004). While such solution ensures that the extracted sentence is correct and meaningful, it is often only loosely related to all relevant terms, and may contain extraneous information. Moreover, such a solution offers little insight on how language generation works, and does not offer the opportunity to incrementally improve our scientific understanding of the intricate mechanisms that direct language generation. In contrast, the approach I advocate builds sentences from scratch, enriching the relevant term list with word dependencies extracted from the input documents, and using stochastic models of language to drive the generation process. A possible output created in this fashion reads "Rain front on Yangtze River valley reaches Wuhan".

I support this language generation paradigm with a novel representation formalism and algorithms that implement a generic NLG system specifically designed for text-to-text applications. The proposed formalism differs significantly from the representation formalisms previously used by generic NLG systems, and it has its origin in the IDL-expression formalism proposed by Nederhof and Satta (2004a).

The formalism of WIDL-expressions (WIDL stands for Weighted Interleave ($\|$), Disjunction ($\vee$), and Lock ($\times$), after the names of its main operators) is an application-independent, information-slim representation language, specifically designed for text-to-text natural language generation. It is a formally defined language, with a simple syntax (expressions are built using four operators, the aforementioned $\|$, $\vee$, $\times$, plus the precedence operator $(\cdot)$) and a simple, formal semantics (probability distributions over finite sets of strings). It has no built-in biases regarding the types of linguistic relations that hold between its atomic elements (such as semantic or syntactic type of relations), which is why I call it an information-slim representation. The advantage of having the representation agnostic to any type of language theory is that it is easy to automatically create WIDL-expressions using information extracted from a single sentence, multiple sentences, or even multiple documents. Biases representing the constraints of particular language theories can be integrated in the generation process after the WIDL representation is created, in a mathematically sound fashion. An additional advantage is that the granularity level is not pre-defined, and, therefore, different levels of granularity (word, phrase, sentence) are handled within the same framework, and the appropriate level is chosen depending on the application itself.

The WIDL formalism is also designed such that true probabilistic models can be described accurately and completely, and therefore the preference integration problem is solved in a principled way. The approach to generation that uses WIDL-expressions is not a hybrid approach anymore, in the sense that statistical knowledge is used both to (over)generate a large set of

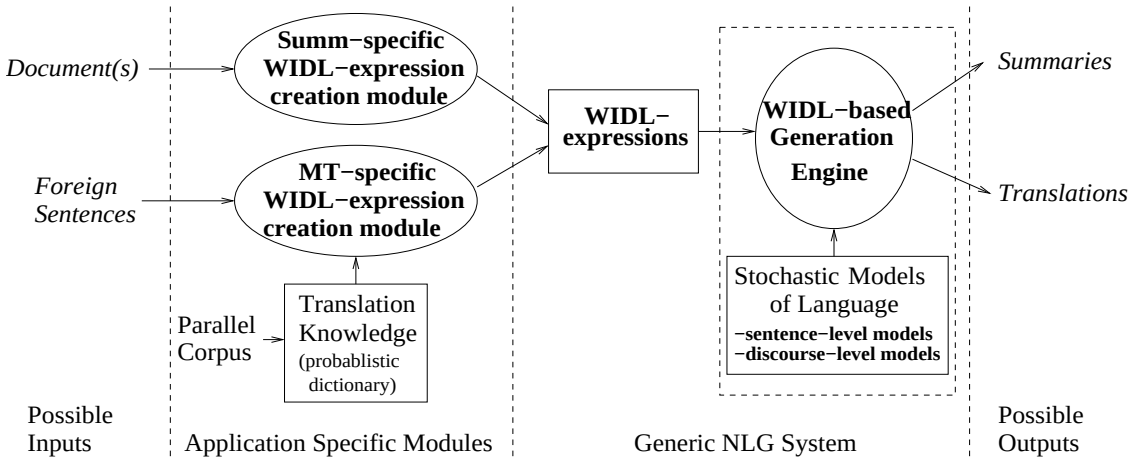

Figure 1.1: A generic, WIDL-based generation engine is used as a back-end module for different text-to-text applications, such as summarization and machine translation. The components in bold font indicate instances in which the original contributions of this dissertation have been deployed.

candidate realizations, and to rank the candidate realizations and find the best scoring one. Although the approach is purely statistical, the representation is flexible enough to be able to express symbolic knowledge when it is so desired (for example, using the precedence operator, one can express that, in English, a determiner appears before the noun it determines).

The approach to generation advocated in this dissertation is summarized in Figure 1.1. A generic, WIDL-based realization engine performs the generation task, driven by both statistical knowledge encapsulated in WIDL-expressions (representing source bias), and statistical knowledge encapsulated in language models (representing target-language bias). Different application-specific modules create WIDL-expressions specific to the task, using external sources of knowledge (such as probabilistic dictionaries for machine translation). For

instance, a summarization application that produces headlines can be implemented by a front-end headline-specific module, followed by the back-end generation module that uses sentence-level language models to produce headline-style summaries (Chapter 4). As another example, a machine translation application can be implemented by a front-end MT-specific module, which creates WIDL-expressions that encapsulate translation-specific knowledge, followed by the back-end generation system that uses stochastic, sentence-level language models such as n-grams to produce translations (Chapter 5). Another instance of the summarization task, called multi-document summarization, requires one to automatically produce a summary text from a collection of input documents. This instance can be implemented using the same architecture, by a front-end module specialized in selecting important, information-bearing units from the input documents to create WIDL-expressions, while the back-end generation engine uses discourse-level models of language to create a summary (Chapter 6).

Sharing a back-end generation engine across multiple applications has the advantage that progress in generic generation capabilities has an immediate impact on the end-to-end performance of all applications that use this architecture. Additionally, one can use the performance on end-to-end applications to measure the degree of progress achieved by using different knowledge sources or different search techniques employed by the generation engine, therefore validating extrisically their usefulness.

1.3 The Usefulness of the WIDL-based Approach to Generation

As the evaluations carried on in this dissertation show, the WIDL-based generation framework is powerful enough to create applications that perform at levels of accuracy similar or surpassing state-of-the-art performance.

For headline generation, the WIDL-based approach produces abstractive headlines at levels of accuracy measured to be comparable (slightly better) with a state-of-the-art, extractive approach, while outperforming previously-proposed abstractive approaches by a wide margin (Chapter 4). For machine translation, the performance of the WIDL-based approach is measured to be comparable (slightly worse) with the performance of a state-of-the-art, phrase-based translation system (Chapter 5). For a coherent discourse generation task, the WIDL framework produces results measured to be significantly better than current state-of-the-art performance (Chapter 7).

Additional evaluations carried on in this framework allow us to measure the contribution of a hierarchic, syntax-based language model to a generation task, compared to the baseline performance of a flat, n-gram–based language model (Chapter 8). This contribution can be measured directly, simply by changing the stochastic language model used by the generation engine. The results of the evaluations performed to measure this contribution indicate that a language model based on a generative model of syntax improves generation accuracy over a baseline trigram language model when the generated sentences are short, but does not currently offer significant improvements when generating long sentences.

1.4 The Contributions of this Dissertation

This dissertation advances the state of the art in several research areas. (The components in bold font in Figure 1.1 indicate instances in which the original contributions of this dissertation have been deployed in actual system implementations.)

Theoretical Contributions to Formal Languages. In this dissertation, I formally specify the syntax and semantics of WIDL-expressions and WIDL-graphs, two equivalent formal languages that represent compactly non-trivial probability distributions over finite, but very large sets of strings. For some of the experiments I conducted, I have created WIDL-expressions and WIDL-graphs that represent as many as 10^{100} strings[1].

Theoretical Contributions to Algorithms. I have designed and implemented algorithms that use the WIDL formalism to efficiently search, score, and rank finite sets of strings encoded as WIDL-expressions, in conjunction with externally-specified probability distributions. These algorithms come with theoretical guarantees about their optimality and efficiency. To be able to search through the large spaces encoded, these algorithms use novel techniques such as admissible estimates for WIDL-expressions and p-admissible estimates for strings.

Empirical Contributions to Algorithm Evaluation. The algorithm evaluations I conduct in this dissertation allow one to empirically validate the advantages, measured in time and space savings, of using WIDL-based algorithms for language generation.

[1] To provide a meaningful comparison for such a large number, I mention here that the number of atoms in the whole Universe is estimated to be around 10^{80}.

Empirical Contributions to Model Evaluation. The model evaluations I conduct in this dissertation allow one to empirically validate the discriminative power of various stochastic models of language for the task of language generation, both at sentence level and at discourse level.

Empirical Contributions to State-of-the-art Performance. The end-to-end application evaluations I conduct in this dissertation show that WIDL-based applications have advanced state-of-the-art performance in abstractive headline generation and coherent discourse generation.

1.5 Dissertation Outline

This dissertation is organized as follows. In Chapter 2, I present the WIDL-expression and WIDL-graph formal languages, together with theoretical results that guarantee their representation properties. In Chapter 3, I present search algorithms that use the WIDL formalism, together with theoretical results concerning the optimality and run-time behavior of these algorithms.

In Chapter 4, I describe and evaluate an end-to-end headline generation system, which uses the WIDL formalism and the proposed generation algorithms to automatically produce headlines for input documents. Chapter 5 is also application-oriented – in it, I describe and evaluate a Machine Translation system for Chinese-English, using the WIDL framework.

In Chapters 6 and 7, I show how the WIDL formalism can be used for generation at discourse level. In conjunction with stochastic models of text coherence, WIDL-expressions are used by the proposed generation algorithms to create texts that exhibit coherence properties.

In Chapter 8, I propose another novel search algorithm, which can be used to evaluate WIDL-expressions in conjunction with hierarchical language models.

Finally, Chapter 9 is a review of the previous work on natural language generation, while Chapter 10 provides conclusions and pointers for future work that arises from the work presented.

Chapter 2

The WIDL Representation Language

In this chapter, I introduce the formalism of WIDL-expressions. WIDL stands for Weighted Interleave, Disjunction, and Lock, after the names of the main operators used in the formalism. WIDL-expressions are proposed with the explicit goal of providing a representation formalism that facilitates the integration of a generic natural language generation system within end-to-end language applications.

The WIDL formalism, an extension of the IDL-expressions formalism proposed by Nederhof and Satta (2004a), has several crucial properties that differentiate it from previously proposed NLG representation formalisms. First, it has a simple syntax (expressions are built using four operators) and a simple, formal semantics (probability distributions over finite sets). Second, it is a compact representation that grows linearly in the number of words available for generation. In contrast, representations such as word lattices (Knight & Hatzivassiloglou 1995) or non-recursive CFGs (Langkilde-Geary 2002) require exponential space in the number of words available for generation (Nederhof & Satta 2004a). Third, it allows for a tight integration of input-specific preferences and target-language preferences via interpolation of probability

distributions using log-linear models (Chapter 3). Fourth, it has good computational properties, such as optimal algorithms for intersection with n-gram language models (Chapter 3). Fifth, it is flexible with respect to the amount of linguistic processing required to produce WIDL-expressions directly from text (Chapters 4 and 5).

2.1 WIDL-expressions

In this section, I introduce WIDL-expressions, a formal language used to *compactly* represent probability distributions over finite sets of strings. First, I provide the intuitions about the WIDL operators and their semantics. Next, I provide a complete formal definition for the class of WIDL-expressions, together with the associated formal semantics.

Given a finite alphabet of symbols Σ, atomic WIDL-expressions are of the form a, with $a \in \Sigma$. For an atomic WIDL-expression $\omega = a$, I define a semantic mapping function that maps ω into a probability distribution, in this case defined as a function $\sigma_{\text{widl}}(\omega) : \mathsf{Dom}_\omega \rightarrow [0, 1]$ defined as $\mathsf{Dom}_\omega = \{a\}$ and $\sigma_{\text{widl}}(\omega)(a) = 1$ (the probability distribution over the singleton set $\{a\}$ which assigns probability 1 to a). Complex WIDL-expressions are created from other WIDL-expressions by employing four operators. Out of the four operators, two (disjunction and interleave) are said to be weighted, as operator distribution functions from a finite alphabet Δ are used to specify probabilities associated with different combinations of the argument expressions. I present these four operators next, together with examples that show their intended meaning.

Weighted Disjunction. If $\omega_1, \ldots, \omega_n$ are WIDL-expressions, then $\omega = \vee_{\delta_0}(\omega_1, \ldots, \omega_n)$, with $\delta_0 : \{1, \ldots, n\} \rightarrow [0, 1]$ specified such that $\Sigma_{a \in \mathsf{dom}(\delta_0)} \delta_0(a) = 1$, is a WIDL-expression. The semantic mapping $\sigma_{\text{widl}}(\omega)$ is a probability distribution defined as $\sigma_{\text{widl}}(\omega) : \mathsf{Dom}_\omega \rightarrow [0, 1]$, where Dom_ω is the set of all strings encoded by $\omega_1, \ldots, \omega_n$, and the probability values are induced by the functions δ_0 and $\sigma_{\text{widl}}(\omega_i)$, $1 \leq i \leq n$. For example, if $\omega = \vee_{\delta_0}(\mathrm{a}, \mathrm{b})$, $\delta_0 = \{1 \rightarrow 0.8,\ 2 \rightarrow 0.2\}$, it represents a probability distribution $\sigma_{\text{widl}}(\omega)$ over the set $\mathsf{Dom}_\omega = \{\mathrm{a}, \mathrm{b}\}$, defined by $\sigma_{\text{widl}}(\omega)(\mathrm{a}) = \delta_0(1) = 0.8$ and $\sigma_{\text{widl}}(\omega)(\mathrm{b}) = \delta_0(2) = 0.2$.

Precedence. If ω_1, ω_2 are WIDL-expressions, then $\omega = \omega_1 \cdot \omega_2$ is a WIDL-expression. The semantic mapping $\sigma_{\text{widl}}(\omega)$ is a probability distribution defined as $\sigma_{\text{widl}}(\omega) : \mathsf{Dom}_\omega \rightarrow [0, 1]$, where Dom_ω is the set of all strings that obey the precedence imposed over the arguments, and the values associated with Dom_ω strings are induced by the distributions $\sigma_{\text{widl}}(\omega_1)$ and $\sigma_{\text{widl}}(\omega_2)$. For example, if $\omega_1 = \vee_{\delta_1}(\mathrm{a}, \mathrm{b})$, $\delta_1 = \{1 \rightarrow 0.8,\ 2 \rightarrow 0.2\}$, and $\omega_2 = \vee_{\delta_2}(\mathrm{c}, \mathrm{d})$, $\delta_2 = \{1 \rightarrow 0.6,\ 2 \rightarrow 0.4\}$, then $\omega = \omega_1 \cdot \omega_2$ represents a probability $\sigma_{\text{widl}}(\omega)$ distribution over the set $\mathsf{Dom}_\omega = \{\mathrm{ac}, \mathrm{ad}, \mathrm{bc}, \mathrm{bd}\}$, defined by $\sigma_{\text{widl}}(\omega)(\mathrm{ac}) = \delta_1(1)\delta_2(1) = 0.48$, $\sigma_{\text{widl}}(\omega)(\mathrm{ad}) = \delta_1(1)\delta_2(2) = 0.32$, $\sigma_{\text{widl}}(\omega)(\mathrm{bc}) = \delta_1(2)\delta_2(1) = 0.12$, and $\sigma_{\text{widl}}(\omega)(\mathrm{bd}) = \delta_1(2)\delta_2(2) = 0.08$.

Weighted Interleave. If $\omega_1, \ldots, \omega_n$ are WIDL-expressions, then $\omega = \|_{\delta_0}(\omega_1, \ldots, \omega_n)$, with $\delta_0 : S \cup \{\mathsf{other\ perms}\} \cup \{\mathsf{shuffles}\} \rightarrow [0, 1]$, $S \subseteq \mathsf{Perm}_n$, specified such that $\Sigma_{a \in \mathsf{dom}(\delta_0)} \delta_0(a) = 1$, is a WIDL-expression. The semantic mapping $\sigma_{\text{widl}}(\omega)$ is a probability distribution defined as $\sigma_{\text{widl}}(\omega) : \mathsf{Dom}_\omega \rightarrow [0, 1]$, where Dom_ω consists of all the possible interleavings of strings from $\mathsf{dom}(\sigma_{\text{widl}}(\omega_i))$, $1 \leq i \leq n$. The probability values are induced by the distributions

$$
\begin{aligned}
&\|_{\delta_1}\ (\ \times(\mathsf{turkish}\cdot\mathsf{government}),\\
&\qquad \vee_{\delta_2}(\times(\mathsf{rebels}\cdot\mathsf{fighting})), \times(\mathsf{attacked}\cdot\mathsf{rebels})),\\
&\qquad \mathsf{in}\cdot\mathsf{iraq}\ )\\
&\delta_1 = \{2\,1\,3 \rightarrow 0.2,\ \mathsf{other\ perms} \overset{\mathsf{uniform}}{\rightarrow} 0.7,\ \mathsf{shuffles} \overset{\mathsf{uniform}}{\rightarrow} 0.1\}\\
&\delta_2 = \{1 \rightarrow 0.65,\ 2 \rightarrow 0.35\}
\end{aligned}
$$

Figure 2.1: Example of WIDL-expression.

$\sigma_{\mathrm{widl}}(\omega_i)$ together with δ_0, defined either explicitly over $S \subseteq \mathsf{Perm}_n$ (the set of all permutations of n elements), or implicitly as $\delta_0(\mathsf{other\ perms})$. As the set of argument permutations is a subset of all possible interleavings, δ_0 also needs to specify the probability mass for the strings that are not argument permutations, $\delta_0(\mathsf{shuffles})$. For example, if $\omega = \|_{\delta_0}(\mathrm{a}\cdot\mathrm{b},\mathrm{c})$, $\delta_0 = \{1\ 2 \rightarrow 0.80,\ \mathsf{other\ perms} \rightarrow 0.15, \mathsf{shuffles} \rightarrow 0.05\}$, it represents a probability distribution $\sigma_{\mathrm{widl}}(\omega)$ with domain $\mathsf{Dom}_\omega = \{\mathrm{abc}, \mathrm{cab}, \mathrm{acb}\}$, defined by $\sigma_{\mathrm{widl}}(\mathrm{abc}) = \delta_0(1\ 2) = 0.80$, $\sigma_{\mathrm{widl}}(\mathrm{cab}) = \delta_0(\mathsf{other\ perms}) = 0.15$, $\sigma_{\mathrm{widl}}(\mathrm{acb}) = \delta_0(\mathsf{shuffles}) = 0.05$.

Lock. If ω' is a WIDL-expression, then $\omega = \times(\omega')$ is a WIDL-expression. The semantic mapping $\sigma_{\mathrm{widl}}(\omega)$ is the same as $\sigma_{\mathrm{widl}}(\omega')$, except that $\mathsf{dom}(\sigma_{\mathrm{widl}}(\omega))$ now contains strings in which no additional symbol can be interleaved. For example, if $\omega = \|_{\delta_0}(\times(\mathrm{a}\cdot\mathrm{b}),\mathrm{c})$, $\delta_0 = \{1\,2 \rightarrow 0.80,\ \mathsf{other\ perms} \rightarrow 0.20\}$, it represents a probability distribution with domain $\mathsf{Dom}_\omega = \{\mathrm{cab}, \mathrm{abc}\}$, defined by $\sigma_{\mathrm{widl}}(\mathrm{abc}) = \delta_0(1\ 2) = 0.80$, $\sigma_{\mathrm{widl}}(\mathrm{cab}) = \delta_0(\mathsf{other\ perms}) = 0.20$., while $\mathrm{acb} \notin \mathsf{Dom}_\omega$.

In Figure 2.1, I show a more complex WIDL-expression. The probability distribution δ_1 associated with the operator $\|_{\delta_1}$ assigns probability 0.2 to the argument order $2\,1\,3$; from a probability mass of 0.7, it assigns uniformly, for each of the remaining $3! - 1 = 5$ argument

permutations, a permutation probability value of $\frac{0.7}{5} = 0.14$. The remaining probability mass of 0.1 is left for the 12 shuffles associated with the unlocked expression $\mathsf{in} \cdot \mathsf{iraq}$, for a shuffle probability of $\frac{0.1}{12} = 0.008$. The list below enumerates some of the $\langle \mathsf{string}, p(\mathsf{string}) \rangle$ pairs that belong to the probability distribution defined by our example:

rebels fighting turkish government in iraq $0.130 = 0.20_{\delta_1} \times 0.65_{\delta_2}$

in iraq attacked rebels turkish government $0.049 = 0.14_{\delta_1} \times 0.35_{\delta_2}$

in turkish government iraq rebels fighting $0.005 = 0.008_{\delta_1} \times 0.65_{\delta_2}$

The following definitions formalize the intuitions provided above. The casual reader may want to skip the formalisms at a first reading, as the overall presentation is intended not to make heavily use of them. This formal fragment covers the definition for the syntax of WIDL-expressions, the formal semantics of WIDL-expressions in terms of probability distributions over finite sets of strings, and a lemma that provides a connection between the formalism of WIDL-expressions and the formalism of IDL-expressions (Nederhof & Satta 2004a).

2.1.1 The Formal Syntax of WIDL-expressions

The class of WIDL-expressions is defined as follows:

Definition 1 *Let Σ be some finite alphabet (called input alphabet), let Δ be another, disjoint, countably infinite alphabet (called distribution function alphabet), and let $\mathcal{E}$ be a symbol not in Σ. A* **WIDL-expression** *over Σ is a string ω satisfying one of the following conditions:*

1. $\omega = a$, *with* $a \in \Sigma \cup \{\mathcal{E}\}$;

2. $\omega = \omega_1 \cdot \omega_2$, *with* ω_1 *and* ω_2 *WIDL-expressions;*

3. $\omega = \times(\omega_1)$, *with* ω_1 *a WIDL-expression.*

4. $\omega = \vee_\delta(\omega_1, \ldots, \omega_n)$, $\delta \in$ *distr_spec*$(\mathcal{G}_{\vee_n})$, *where* *distr_spec*$(\mathcal{G}_{\vee_n})$ *is the language specified by grammar* $\mathcal{G}_{\vee_n}$, *and* ω_i *a WIDL-expression,* $1 \leq i \leq n$. *Grammar* $\mathcal{G}_{\vee_n} = \langle \mathcal{NT}_{\vee_n}, \mathcal{T}_{\vee_n}, D_n, \mathcal{P}_{\vee_n} \rangle$ *has the set of nonterminals* $\mathcal{NT}_{\vee_n} = \{D_n, A^p_{i,n} \mid 1 \leq i \leq n, p \in [0,1]\}$, *the set of terminal symbols* $\mathcal{T}_{\vee_n} = \{i \rightarrow p \mid 1 \leq i \leq n, p \in [0,1]\}$, D_n *as starting symbol, and the following production set* $\mathcal{P}_{\vee_n}$:

$$
\begin{array}{lcll}
D_n & \rightarrow & A^1_{1,n} & \\
A^p_{i,n} & \rightarrow & i \rightarrow p_i \; A^{p-p_i}_{i+1,n} & \text{if } i < n, \text{ and } p, p_i \in [0,1] \\
A^p_{n,n} & \rightarrow & n \rightarrow p_n & \text{if } p, p_n \in [0,1]
\end{array}
$$

5. $\omega = \|_\delta(\omega_1, \ldots, \omega_n)$, $\delta \in$ *distr_spec*$(\mathcal{G}_{\|_n})$, *where* *distr_spec*$(\mathcal{G}_{\|_n})$ *is the language specified by grammar* $\mathcal{G}_{\|_n}$, *and* ω_i *a WIDL-expression,* $1 \leq i \leq n$. *Grammar* $\mathcal{G}_{\|_n} = \langle \mathcal{NT}_{\|_n}, \mathcal{T}_{\|_n}, I_n, \mathcal{P} \rangle$ *has the set of nonterminals* $\mathcal{NT}_n = \{I_n, L^p_S, O^p, Sh^p \mid S \subseteq \mathsf{Perm}_n, p \in [0,1]\}$ *(where* Perm_n *is the set of all permutations over elements 1,*$\ldots, n$*), the set of terminal symbols* $\mathcal{T}_n = \{perm_i \rightarrow p_k$, perms $\xrightarrow{f} p_k$, other perms $\xrightarrow{f} p_k$, shuffles $\xrightarrow{\text{uniform}} p_k \mid perm_i \in \mathsf{Perm_n}, S \subseteq \mathsf{Perm}_n, f \in \{\text{uniform}, \text{exp\_mov\_pen}\}$, p_k *symbols for reals from* $[0,1]\}$, I_n *as starting symbol, and the following production set* $\mathcal{P}_{\|_n}$:

$$
\begin{array}{lcll}
I_n & \rightarrow & L^1_{\mathsf{Perm}_n} & \\
L^p_S & \rightarrow & perm_i \rightarrow p_i \; L^{p-p_i}_{S\setminus\{perm_i\}} \mid O^p \mid Sh^p & \textit{if } p, p_i \in [0,1], \textit{and } perm_i \in S \\
O^1 & \rightarrow & \mathsf{perms} \xrightarrow{f} p \;\; Sh^{1-p} & \textit{if } p \in [0,1], \\
 & & & f \in \{\mathsf{uniform}, \mathsf{exp\_mov\_pen}\} \\
O^p & \rightarrow & \mathsf{other\ perms} \xrightarrow{f} p_o \; Sh^{p-p_o} & \textit{if } p, p_o \in (0,1), \\
 & & & f \in \{\mathsf{uniform}, \mathsf{exp\_mov\_pen}\} \\
O^0 & \rightarrow & \epsilon & \\
Sh^p & \rightarrow & \mathsf{shuffles} \xrightarrow{\mathsf{uniform}} p & \textit{if } p \in (0,1] \\
Sh^0 & \rightarrow & \epsilon &
\end{array}
$$

The distribution functions δ associated with an n-ary $\vee$ operators are written simply by specifying the values associated with each of the n choices, respecting the order of the arguments. The distribution functions δ associated with an n-ary $||$ operator can be written either explicitly, by assigning to a permutation $perm_i$ a value p_i, or implicitly, by using a symbol $f \in \{\mathsf{uniform}, \mathsf{exp\_mov\_pen}\}$. Note that it is not our intention to restrict these symbols to only uniform and exp_mov_pen. Rather, the formal semantics of WIDL-expressions is given (at least for now) by assigning interpretations only to these two symbols, as they are the ones used in the applications of WIDL-expressions presented in this thesis. For other applications, it might be necessary to extend the formalism to other symbols and interpretations of these symbols, for the purpose of assigning implicitly probability values to permutations and shuffles.

2.1.2 The Formal Semantics of WIDL-expressions

The semantics of WIDL-expressions is given in terms of probability distributions over finite sets of strings. The domains of these probability distributions are defined using an

interpretation of the $\cdot, \times, \vee$, and $\|$ operators similar with the one given in (Nederhof & Satta 2004a). In addition, I define the interpretation of the operator distribution functions, which allows one to assign probability values to each string from the defined domains.

The semantic interpretation uses, for the proper treatment of the $\times$ operator, a new symbol $\diamond$, not in Σ. An occurrence of $\diamond$ between two strings indicates that it is allowed for an additional string to be inserted at that position. The $\times$ operator corresponds to the removal of every occurrence of the $\diamond$ symbol. More precisely, there exists a homomorphism $\texttt{lock}$ from the free monoid of strings over $(\Sigma \cup \{\diamond\})^*$ to the free monoid of strings over Σ^*, defined as the unique extension of the mappings $\texttt{lock}(a) = a$, $a \in \Sigma$ and $\texttt{lock}(\diamond) = \epsilon$. Applying $\texttt{lock}$ to an arbitrary string over $(\Sigma \cup \{\diamond\})^*$ has the effect of concatenating together all the strings in Σ^* from that sequence. Another operation, $\texttt{comb}$, is needed to express all the possible interleavings of two string sequences. The definition of $\texttt{comb}$ is the following:

$$\texttt{comb}(x, y) = \texttt{comb}'(x, y) \cup \texttt{comb}'(y, x)$$

$$\texttt{comb}'(x, y) = \begin{cases} \{x \diamond y\} & \text{if } x \in \Sigma^* \\ \{x' \diamond\} \cdot \texttt{comb}(x'', y) & \text{if } x = x' \diamond x'', x' \in \Sigma^* \end{cases}$$

Operation $\texttt{comb}$ uses an auxiliary operation $\texttt{comb}'$, which also constructs interleaved sequences from input strings x and y, but always starting with the first string of its first argument x. As any sequence in $\texttt{comb}(x, y)$ must start with a string from x or a string from y, $\texttt{comb}(x, y)$ is the union of $\texttt{comb}'(x, y)$ and $\texttt{comb}'(y, x)$. The definition of $\texttt{comb}'$ distinguishes the case when x is a string in Σ^* (i.e., no $\diamond$ symbol, which means no interleave possible), and the case in which x consists of at least two strings in Σ^*, in which case the

tail of an output sequence is obtained by applying comb recursively on the tail of x and the entire sequence y.

The definition of comb is extended to languages L_1, L_2 as follows:

$$\mathtt{comb}(L_1, L_2) = \bigcup_{x \in L_1, y \in L_2} \mathtt{comb}(x, y),$$

and generalized to n languages by $\mathtt{comb}_{i=1}^{n} L_i = \mathtt{comb}(L_1, L_2)$ for $n = 2$, and $\mathtt{comb}_{i=1}^{n} L_i = \mathtt{comb}(\mathtt{comb}_{i=1}^{n-1}(L_i, L_n))$ for $n > 2$.

I also define here the interpretation of the symbols used by $\mathcal{G}_{\|_n}$, uniform and exp_mov_pen, as functions. Function uniform is used to assign the same probability value uniformly. Function exp_mov_pen is used to assign higher values to permutation orders that are closer to the monotonic order, by penalizing movement exponentially using a distance measure based on index offsets.

$$\mathsf{uniform} : \mathbf{N} \to [0,1] \quad \mathsf{uniform}(n) = \tfrac{1}{n}$$

$$\mathsf{exp\_mov\_pen} : S \subseteq \mathsf{Perm}_m \to [0,1] \quad \mathsf{exp\_mov\_pen}(q_1 \ldots q_n) = \frac{\exp(-\Sigma_{i=1}^{n} \mathsf{d}(q_i))}{\Sigma_{q' \in S} \exp(-\Sigma_{i=1}^{n} \mathsf{d}(q'_i))},$$

$$\text{where } \mathsf{d}(q_i) = \mathsf{abs}(\mathsf{index}(q_i - 1) + 1 - \mathsf{index}(q_i))$$

For a sequence such as $q_1\ q_2\ q_3 = 2\ 3\ 1$, one has $\mathsf{index}(q_1\ q_2\ q_3) = 1\ 2\ 3$, and function d yields the values:

$$\mathsf{d}(2) = \mathsf{abs}(\mathsf{index}(2-1) + 1 - \mathsf{index}(2)) = \mathsf{abs}(3 + 1 - 1) = 3$$

$$\mathsf{d}(3) = \mathsf{abs}(\mathsf{index}(3-1) + 1 - \mathsf{index}(3)) = \mathsf{abs}(1 + 1 - 2) = 0$$

$$\mathsf{d}(1) = \mathsf{abs}(\mathsf{index}(1-1) + 1 - \mathsf{index}(1)) = \mathsf{abs}(0 + 1 - 3) = 2$$

for a total $\Sigma_{i=1}^{n} \mathsf{d}(q_i) = 5$. The numerator is therefore $\exp(-5) = 0.0067$, while the denominator (summing over all index sequences of length 3) is 1.127. Therefore, we have

that $\mathsf{exp\_mov\_pen}(2\ 3\ 1) = \frac{0.0067}{1.127} = 0.006$, while for the monotone sequence 1 2 3 gives $\mathsf{exp\_mov\_pen}(1\ 2\ 3) = \frac{1}{1.127} = 0.887$.

I provide now the formal definition for the semantics of WIDL-expressions.

Definition 2 *Let Σ be some finite alphabet (called input alphabet), let Δ be another, disjoint, countably infinite alphabet (called distribution function alphabet). Function $\sigma_{\mathrm{widl}}(\omega) : \mathrm{Dom}_\omega \rightarrow [0,1]$ is the probability distribution associated with WIDL-expression ω, where:*

1. *if $\omega = a$, then $\mathrm{Dom}_\omega = \{a\}$, $\sigma_{\mathrm{widl}}(\omega)(a) = 1$, for any $a \in \Sigma \cup \{\mathcal{E}\}$;*
2. *if $\omega = \omega_1 \cdot \omega_2$, then $\mathrm{Dom}_\omega = \mathrm{Dom}_{\omega_1} \diamond \mathrm{Dom}_{\omega_2}$, $\sigma_{\mathrm{widl}}(\omega)(e_1 \diamond e_2) = \sigma_{\mathrm{widl}}(\omega_1)(e_1) \cdot \sigma_{\mathrm{widl}}(\omega_2)(e_2)$, for any $e_1 \in \mathrm{Dom}_{\omega_1}$ and $e_2 \in \mathrm{Dom}_{\omega_2}$;*
3. *if $\omega = \times(\omega_1)$, then $\mathrm{Dom}_\omega = \mathtt{lock}(\mathrm{Dom}_{\omega_1})$, $\sigma_{\mathrm{widl}}(\omega)(e) = \sigma_{\mathrm{widl}}(\omega_1)(e_1)$, for any $e = \mathtt{lock}(e_1)$, $e_1 \in \mathrm{Dom}_{\omega_1}$;*
4. *if $\omega = \vee_{\delta_0}(\omega_1, \ldots, \omega_n), \delta_0 = \{1 \rightarrow p_1, \ldots, n \rightarrow p_n\}$, then $\mathrm{Dom}_\omega = \cup_{i=1}^n \mathrm{Dom}_{\omega_i}$, $\sigma_{\mathrm{widl}}(\omega)(e) = p_i \cdot \sigma_{\mathrm{widl}}(\omega_i)(e)$, for any $e \in \mathrm{Dom}_{\omega_i}$, $1 \leq i \leq n$;*
5. *if $\omega = \|_{\delta_0}(\omega_1, \ldots, \omega_n), \delta_0 = \{perm \rightarrow p, \mathsf{other\ perm} \xrightarrow{f} p_o, \mathsf{shuffles} \xrightarrow{\mathsf{uniform}} p_s\}$, $f \in \{\mathsf{uniform}, \mathsf{exp\_mov\_pen}\}$, then $\mathrm{Dom}_\omega = \mathtt{comb}_{i=1}^n \mathrm{Dom}_{\omega_i}$, and, if $N_o = n! - 1$ and, for $\omega_\times = \|_{\delta_0}(\times(\omega_1), \ldots, \times(\omega_n)), N_s = |\mathrm{Dom}_\omega| - |\mathrm{Dom}_{\omega_\times}|$, then*

$$\sigma_{\mathrm{widl}}(\omega)(e) = \begin{cases} p \cdot \Pi_{k=1}^n \sigma_{\mathrm{widl}}(\omega_{i_k}) & \text{if } i_1 \ldots i_n = perm \\ \mathsf{uniform}(N_o) \cdot p_o \cdot \Pi_{k=1}^n \sigma_{\mathrm{widl}}(\omega_{i_k}) & \text{if } i_1 \ldots i_n \neq perm \\ & \text{and } f = \mathsf{uniform} \\ \mathsf{exp\_mov\_pen}(i_1 \ldots i_n) \cdot p_o \cdot \Pi_{k=1}^n \sigma_{\mathrm{widl}}(\omega_{i_k}) & \text{if } i_1 \ldots i_n \neq perm \\ & \text{and } f = \mathsf{exp\_mov\_pen} \end{cases}$$

for any $e = e_{i_1} \ldots e_{i_n}, e_{i_j} \in \mathrm{Dom}_{\omega_{i_j}}, 1 \leq j \leq n$, *and* $\sigma_{\mathrm{widl}}(\omega)(e) = \mathsf{uniform}(N_s) \cdot p_s$, *for any other* $e \in \mathrm{Dom}_{\omega}$.

Note that, for the clarity of the exposition, I have only provided the definition for $\sigma_{\mathrm{widl}}(\omega)$ for ω expressions for which the distribution function of the $\|$ operator specifies only one explicit permutation. In this case, the number of implicitly specified permutations of n indexes, N_o, is $n! - 1$. The definition extends trivially to the case when more than one explicit permutation (or none) is specified. The total number of shuffles is given by the difference between the size of the domain of the probability distribution of the ω expression and the the size of the domain of the distribution specified by the expression $\omega_{\times}$, which has each argument of the $\|$ operator locked. As no shuffles can occur for the expression $\omega_{\times}$, the difference is precisely the number of shuffled strings for the expression ω.

The last item in this formal fragment is a lemma that makes explicit the connection between the formalism of IDL-expressions (Nederhof & Satta 2004a) and the formalism of WIDL-expressions.

Lemma 1 *For any WIDL-expression* ω *over finite alphabets* Σ *and* Δ, *there exists an IDL-expression* π *over* Σ *such that* $\mathsf{dom}(\sigma_{\mathrm{widl}}(\omega)) = \sigma_{\mathrm{idl}}(\pi)$. *Conversely, for any IDL-expression* π *over* Σ, *there exists a WIDL-expression* ω *over* Σ *and* Δ, *such that* $\sigma_{\mathrm{idl}}(\pi) = \mathsf{dom}(\sigma_{\mathrm{widl}}(\omega))$.

Lemma 1 follows from Definition 2 for σ_{widl} and the definition of σ_{idl} (Nederhof & Satta 2004a). The implication is that, modulo the specification of operator distributions (such as δ_1 and δ_2 in Figure 2.1), the representation properties of IDL-expressions carry over to WIDL-expressions.

The following theorem provides an important characterization of the representation properties of the WIDL-formalism.

Theorem 2 *A WIDL-expression ω over a finite alphabet Σ using n atomic expressions has space complexity $\mathrm{O}(n)$, if the operator distribution functions of ω have space complexity at most $\mathrm{O}(n)$.*

Theorem 2 follows directly from Lemma 1 and the space complexity result for IDL-expressions from (Nederhof & Satta 2004a). In essence, it tells us that if the δ distributions for all $\|_\delta$ operators are linearly specified, then WIDL-expressions grow linearly in size with the number of words available for generation. This property is particularly important for scaling WIDL-expressions to represent high-complexity hypothesis spaces (see Chapter 5).

2.2 WIDL-graphs

Equivalent at the representation level with WIDL-expressions, WIDL-graphs allow for formulations of algorithms that process them. For each WIDL-expression ω, there exists a corresponding WIDL-graph γ_ω. As an example, I illustrate in Figure 2.2 the WIDL-graph corresponding to the WIDL-expression in Figure 2.1. WIDL-graphs have an initial vertex v_s and a final vertex v_e. Vertices v_0, v_6, and v_{20} with in-going edges labeled $\vdash^1_{\delta_1}$, $\vdash^2_{\delta_1}$, and $\vdash^3_{\delta_1}$, respectively, and vertices v_5, v_{19}, and v_{23} with out-going edges labeled $\dashv^1_{\delta_1}$, $\dashv^2_{\delta_1}$, and $\dashv^3_{\delta_1}$, respectively, result from the expansion of the $\|_{\delta_1}$ operator. Vertices v_7 and v_{13} with in-going edges labeled $(^1_{\delta_2}$, $(^2_{\delta_2}$, respectively, and vertices v_{12} and v_{18} with out-going edges labeled $)^1_{\delta_2}$, $)^2_{\delta_2}$, respectively, result from the expansion of the $\vee_{\delta_2}$ operator. Vertices v_1 to v_4, v_8 to v_{11}, and v_{14} to v_{17} result from

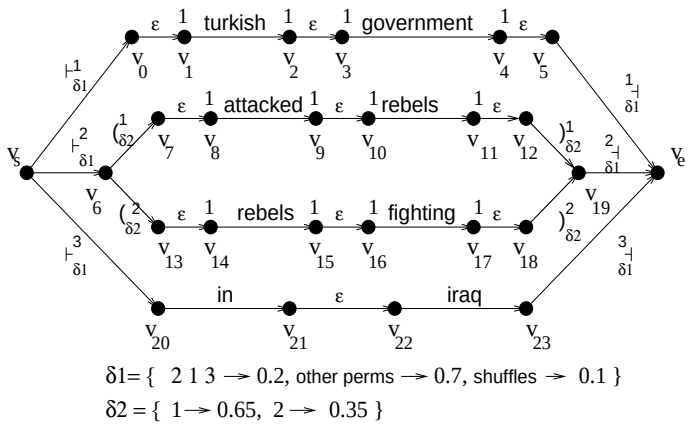

Figure 2.2: The WIDL-graph corresponding to the WIDL-expression in Figure 2.1.

the expansion of the three $\times$ operators, respectively. These latter vertices are also shown to have rank 1, as opposed to rank 0 (not shown) assigned to all other vertices. The ranking of vertices in a WIDL-graph is needed to enforce a higher priority on the processing of the higher-ranked vertices, such that the desired semantics for the lock operator is preserved.

The following formal fragment offers a complete definition of WIDL-graphs. It also offers the formal definition of the mapping between WIDL-expressions and WIDL-graphs.

Let Σ be some finite alphabet (called input alphabet), and let Δ be another, disjoint, countably infinite alphabet (called distribution function alphabet). WIDL-graphs are tuples of the form $(V, E, v_s, v_e, \lambda, r)$, where:

- V and E are finite sets of vertices and edges, respectively;
- v_s and v_e are special vertices in V, called *start* and *end* vertices, respectively;
- λ is the edge-labeling function, mapping E into the alphabet $\Sigma \cup \{\varepsilon, (^i_\delta,)^i_\delta, \vdash^i_\delta, \dashv^i_\delta \mid i > 0,\ \delta \in \Delta\}$;

- r is the vertex-ranking function, mapping V to the set of non-negative integer numbers;
- for every maximal set $\{(^i_\delta,)^i_\delta \mid 1 \leq i \leq n\}$, δ is a string from the language $distr\_spec(\mathcal{G}_{\vee_n})$ specified by grammar $\mathcal{G}_{\vee_n}$;
- for every maximal set $\{\vdash^i_\delta, \dashv^i_\delta \mid 1 \leq i \leq n\}$, δ is a string from the language $distr\_spec(\mathcal{G}_{\|_n})$ specified by grammar $\mathcal{G}_{\|_n}$

Label ε indicates that an edge does not consume an input symbol; labels $\vdash^i_\delta$, $\dashv^i_\delta$, and $(^i_\delta$, $)^i_\delta$ have the same interpretation, plus indicating that it is the start/end of what corresponds to a $\|_\delta$ and $\vee_\delta$ operator, respectively, with superscript i indicating the index of the argument to which the end/start vertex corresponds. Function r ranks each vertex according to how deeply it is embedded into the encoding of expressions headed by $\times$ operators. This information is needed to handle "locked" strings (or, rather, their corresponding subgraphs) with the intended interpretation of the lock ($\times$) operator. Finally, for each δ symbol a specification is given, using the same grammars as used for specifying the distribution functions for WIDL-expressions.

2.2.1 Mapping WIDL-expressions into WIDL-graphs

The next definition provides the formal definition for the mapping of a WIDL-expression into its corresponding WIDL-graph.

Definition 3 *Let Σ be a finite alphabet (called input alphabet), let Δ be another, disjoint, countably infinite alphabet (called distribution alphabet), and let j be a non-negative integer number. Each WIDL-expression ω over Σ and Δ is associated with a graph $\gamma_j(\omega) = (V, E, v_s, v_e, \lambda, r)$, specified as follows:*

1. *if* $\omega = a, a \in \Sigma \cup \{\mathcal{E}\}$, *let* v_s, v_e *be new nodes; I define*

   (a) $V = \{v_s, v_e\}$,

   (b) $E = \{(v_s, v_e)\}$,

   (c) $\lambda((v_s, v_e)) = a$ *for* $a \in \Sigma$ *and* $\lambda((v_s, v_e)) = \varepsilon$ *for* $a = \mathcal{E}$,

   (d) $r(v_s) = r(v_e) = j$;

2. *if* $\omega = \omega_1 \cdot \omega_2$ *with* $\gamma_j(\omega_i) = (V_i, E_i, v_{i,s}, v_{i,e}, \lambda_i, r_i)$, $i \in \{1, 2\}$, *let* v_s, v_e *be new nodes; then*

   (a) $V = V_1 \cup V_2$,

   (b) $E = E_1 \cup E_2 \cup \{(v_{1,e}, v_{2,s})\}$,

   (c) $\lambda(e) = \lambda_i(e)$ *for* $e \in E_1 \cup E_2$, $\lambda((v_{1,e}, v_{2,s})) = \varepsilon$,

   (d) $r(v) = r_i(v)$ *for* $v \in V_1 \cup V_2$;

3. *if* $\omega = \times(\omega')$ *with* $\gamma_{j+1}(\omega') = (V', E', v'_s, v'_e, \lambda', r')$, *let* v_s, v_e *be new nodes; then*

   (a) $V = V' \cup \{v_s, v_e\}$,

   (b) $E = E' \cup \{(v_s, v'_s), (v'_e, v_e)\}$,

   (c) $\lambda(e) = \lambda'(e)$ *for* $e \in E'$, $\lambda((v_s, v'_s)) = \lambda((v'_e, v_e)) = \varepsilon$,

   (d) $r(v) = r'(v)$ *for* $v \in V'$, $r(v_s) = r(v_e) = j$;

4. *if* $\omega = \vee_\delta(\omega_1, \omega_2, \ldots, \omega_n)$ *with* $\gamma_j(\omega_i) = (V_i, E_i, v_{i,s}, v_{i,e}, \lambda_i, r_i)$, $1 \leq i \leq n$, *let* v_s, v_e *be new nodes; then*

   (a) $V = \cup_{i=1}^n V_i \cup \{v_s, v_e\}$,

   (b) $E = \cup_{i=1}^n E_i \cup \{(v_s, v_{i,s}) | 1 \leq i \leq n\} \cup \{(v_{i,e}, v_e) | 1 \leq i \leq n\}$,

   (c) $\lambda(e) = \lambda_i(e)$ *for* $e \in E_i$, $\lambda((v_s, v_{i,s})) = (^i_\delta$, $\lambda((v_{i,e}, v_e)) =)^i_\delta$, $1 \leq i \leq n$,

   (d) $r(v) = r_i(v)$ *for* $v \in V_i$, $r(v_s) = r(v_e) = j$, $1 \leq i \leq n$;

5. *if* $\omega = \|_\delta(\omega_1, \omega_2, \ldots, \omega_n)$ *with* $\gamma_{j+1}(\omega_i) = (V_i, E_i, v_{i,s}, v_{i,e}, \lambda_i, r_i)$, $1 \leq i \leq n$, *let* v_s, v_e *be new nodes; then*

   (a) $V = \cup_{i=1}^{n} V_i \cup \{v_s, v_e\}$,

   (b) $E = \cup_{i=1}^{n} E_i \cup \{(v_s, v_{i,s}), |1 \leq i \leq n\} \cup \{(v_{i,e}, v_e), |1 \leq i \leq n\}$,

   (c) $\lambda(e) = \lambda_i(e)$ *for* $e \in E_i$, $\lambda((v_s, v_{i,s})) = \vdash_\delta^i$, $\lambda((v_{i,e}, v_e)) = \dashv_\delta^i$, $1 \leq i \leq n$,

   (d) $r(v) = r_i(v)$ *for* $v \in V_i$, $r(v_s) = r(v_e) = j$, $1 \leq i \leq n$;

I define $\gamma(\omega) = \gamma_0(\omega)$. *A* **WIDL-graph** *is a graph that has the form* $\gamma(\omega)$ *for some WIDL-expression* ω, *and the same specification for its distribution functions as* ω.

In what follows, I will use the notation γ_ω for $\gamma(\omega)$.

Each WIDL-graph γ_ω has an associated probability distribution, written $\sigma_{\text{widl}}(\gamma_\omega)$. The domain of this distribution is the finite collection of strings that can be generated from a WIDL-specific traversal, or unfolding, of γ_ω, starting from v_s and ending in v_e. Each unfolded path (and its associated string) has a probability value obtained as the product of the probabilities associated with the edge labels of γ_ω.

The next formal fragment defines the semantics of WIDL-graphs in terms of probability distributions over finite strings. The main ingredient in this definition is relation UNFOLD_ω^n, which defines a WIDL-specific traversal, or unfolding, of the graph γ_ω. Relation UNFOLD_ω^n is also crucial for establishing a formal connection between the formalism of WIDL-expressions and the formalism of probabilistic finite-state automata, which is the topic of the next section. Finally, the formal fragment also provides the connection, at a semantic level, between

WIDL-expressions and WIDL-graphs. A WIDL-expression ω and its corresponding WIDL-graph γ_ω are said to be equivalent because they represent the same probability distribution, that is, $\sigma_{\text{widl}}(\gamma_\omega) = \sigma_{\text{widl}}(\omega)$.

First, I introduce some notations used throughout this formal fragment.

Let ω be a fixed WIDL-expression and $\gamma_\omega = (V, E, v_s, v_e, \lambda, r)$ be the associated WIDL-graph. If V is a finite alphabet, then $\hat{V}$ is the set that contains the strings over V in which each symbol occurs at most once. Therefore $\hat{V}$ is a finite set and for each string $c \in \hat{V}$ one has $|c| \leq |V|$. If one considers a linear order on the outgoing edges of each vertex in γ_ω, one can represent cuts in a canonical way by means of strings in $\hat{V}$ as defined below.

Let r be the ranking function associated with γ_ω. I write $c[v_1, \ldots, v_m]$ to denote a string $c \in \hat{V}$ satisfying the following two conditions:

- c has the form $xv_1 \ldots v_m y$ with $x, y \in V$, $1 \leq i \leq m$;
- for each vertex v in c and each $v_i, 1 \leq i \leq m$, we have $r(v) \leq r(v_i)$.

Intuitively, $c[v_1, \ldots, v_m]$ indicate that vertices $v_1, \ldots, v_m$ occur adjacently in c and have the maximal rank among all vertices within string c. One needs to distinguish between vertices with maximal rank from those of lower rank because the maximal ones correspond to WIDL-subexpressions headed by the $\times$ operator that are nested deeper. Because one wants to forbid interleaving strings within the scope of an $\times$ operator with strings outside the scope of the $\times$ operator, one needs a mechanism to enforce the processing of the higher-ranked vertices before resuming the processing of the lower-ranked ones.

For $c[v_1, \ldots, v_m] = xv_1 \ldots v_m y$ in $\hat{V}$, I write $c[v_1 \ldots v_m := v'_1 \ldots v'_{m'}]$ to denote the string $xv'_1 \ldots v'_{m'} y \in \hat{V}$, when no symbol $v'_i, 1 \leq i \leq m'$, appears in x or y. Also, for any

finite alphabet Γ and string $x = x_n x_{n-1} \ldots x_1$ in Γ^*, I define the functions $\mathsf{tail} : \Gamma^* \to \Gamma^*$, $\mathsf{tail}(x) = x_{n-1} \ldots x_1$ and $\mathsf{last} : \Gamma^* \to \Gamma$, $\mathsf{last}(x) = x_1$. By definition, $\mathsf{tail}(\epsilon) = \epsilon$ and $\mathsf{last}(\epsilon) = \epsilon$.

2.2.2 The Unfolding of a WIDL-graph

The first notion I define is a relation, called UNFOLD. This relation is used to define a set, called n-$\mathsf{cut}(\omega)$, and, together, these notions define the traversal, of unfolding, of a WIDL-graph.

Definition 4 *Let Σ be a finite alphabet, ω be a WIDL-expression over Σ and Δ and $\gamma_\omega = (V, E, v_s, v_e, \lambda, r)$ be the associated WIDL-graph. Let n be a natural number, and let Σ^n denote a string over alphabet Σ of length at most n. Let $\mathbf{S} = (\mathbf{N} \cup \{\langle_\delta, \rangle_\delta | \delta \in \Delta\})^*$ denote an arbitrary string over the set of natural numbers and the symbols $\langle_\delta, \rangle_\delta$, for any $\delta \in \Delta$. The relation $\text{UNFOLD}^n_{\gamma_\omega} \subseteq (\hat{V} \times \Sigma^n \times \mathbf{S}) \times (\Sigma \cup \{\varepsilon\}) \times (\hat{V} \times \Sigma^n \times \mathbf{S})$ is the smallest relation satisfying the following conditions:*

1. *for each $c[v] \in \hat{V}$, $(v, v') \in E$ with $\lambda((v, v')) = X \in \Sigma \cup \{\varepsilon\}$, $h \in \Sigma^n$, and $s \in \mathbf{S}$, $\mathsf{last}(s) \in \mathbf{N} \cup \{\epsilon\}$, then*

$$(C, X, C') \in \text{UNFOLD}^n_{\gamma_\omega} \quad \textit{for} \quad C = (c[v], h, s), \quad C' = (c[v := v'], \mathsf{tail}(h)X, s), \textit{ and } X \in \Sigma \tag{2.1}$$

$$(C, \varepsilon, C') \in \text{UNFOLD}^n_{\gamma_\omega} \quad \textit{for} \quad C = (c[v], h, s),\ C' = (c[v := v'], h, s)$$

2. *for each* $c[v] \in \hat{V}$ *with the outgoing edges of* v *being exactly* $(v, v_1), \ldots, (v, v_k)$*, and with* $\lambda((v, v_i)) = \vdash^i_{\delta_0}, 1 \leq i \leq k$*, and* $h \in \Sigma^n$*,* $s \in \mathbf{S}$*, then*

$$(C, \varepsilon, C') \in \text{UNFOLD}^n_{\gamma_\omega} \quad \textit{for} \quad C = (c[v], h, s), \qquad (2.2)$$
$$C' = (c[v := v_1 \ldots v_k], h, s\langle_{\delta_0}\rangle_{\delta_0})$$

3. *for each* $c[v] \in \hat{V}$ *and* $(v, v') \in E$ *with* $\lambda((v, v')) = X \in \Sigma \cup \{\varepsilon\}$*, if* $h \in \Sigma^n$*,* $s \in \mathbf{S}$*, and* $i \in \mathbf{N}$*, and* $(v_0, v_1), \ldots, (v_n, v) \in E$ *is minimal such that* $\lambda((v_0, v_1)) = \vdash^i_{\delta_0}$ *then*

$$(C, X, C') \in \text{UNFOLD}^n_{\gamma_\omega} \quad \textit{for} \quad C = (c[v], h, s\rangle_{\delta_0}), \qquad (2.3)$$
$$C' = (c[v := v'], \mathsf{tail}(h)X, si),\ X \in \Sigma$$
$$(C, \varepsilon, C') \in \text{UNFOLD}^n_{\gamma_\omega} \quad \textit{for} \quad C = (c[v], h, s\rangle_{\delta_0}),\ C' = (c[v := v'], h, si)$$

4. *for each* $c[v] \in \hat{V}$ *and* $(v, v') \in E$ *with* $\lambda((v, v')) = \dashv^i_{\delta_0}$*, if* $h \in \Sigma^n$*,* $s \in \mathbf{S}$*, and* $i \in \mathbf{N}$*, and* $(v_0, v_1), \ldots, (v_n, v) \in E$ *is minimal such that* $\lambda((v_0, v_1)) = \vdash^i_{\delta_0}$*, then*

$$(C, \varepsilon, C') \in \text{UNFOLD}^n_{\gamma_\omega} \quad \textit{for} \quad C = (c[v], h, si),\ C' = (c[v], h, si\rangle_{\delta_0}) \quad (2.4)$$

5. *for each* $c[v] \in \hat{V}$ *and* $(v, v') \in E$ *with* $\lambda((v, v')) = X \in \Sigma \cup \{\varepsilon\}$*, if* $h \in \Sigma^n$*,* $s \in \mathbf{S}$*,* $s' \in \mathbf{N}^*$*,* $i \in \mathbf{N}$*, and* $(v_0, v_1), \ldots, (v_n, v) \in E$ *minimal,* $\lambda((v_0, v_1)) = \vdash^j_{\delta_0}, j \neq i$*, then*

$$(C, X, C') \in \text{UNFOLD}^n_{\gamma_\omega} \quad \textit{for} \quad C = (c[v], h, s\langle_{\delta_0} s'i), \qquad (2.5)$$
$$C' = (c[v := v'], \mathsf{tail}(h)X, s\langle_{\delta_0} 0),\ X \in \Sigma$$

$$(C, \varepsilon, C') \in \text{UNFOLD}^n_{\gamma_\omega} \quad \textit{for} \quad C = (c[v], h, s\langle_{\delta_0} s' i),\ C' = (c[v := v'], h, s\langle_{\delta_0} 0)$$

6. *for each* $c[v] \in \hat{V}$ *with the incoming edges of* v *being exactly* $(v_1, v), \ldots, (v_n, v)$, *and with* $\lambda((v_i, v)) = \dashv^i_{\delta_0}, 1 \leq i \leq n$, *and* $h \in \Sigma^n$, $s \in \mathbf{S}$, *then*

$$\begin{aligned} (C, \varepsilon, C') \in \text{UNFOLD}^n_{\gamma_\omega} \quad \textit{for} \quad & C = (c[v_1 \ldots v_n], h, s\langle_{\delta_0} i_1 \ldots i_n\rangle_{\delta_0}), \\ & C' = (c[v_1 \ldots v_n := v], h, s) \\ (C, \varepsilon, C') \in \text{UNFOLD}^n_{\gamma_\omega} \quad \textit{for} \quad & C = (c[v_1 \ldots v_n], h, s\langle_{\delta_0} 0\rangle_{\delta_0}), \\ & C' = (c[v_1 \ldots v_n := v], h, s) \end{aligned} \tag{2.6}$$

Henceforth, the notation UNFOLD^n_ω is used in place of $\text{UNFOLD}^n_{\gamma_\omega}$, and notation UNFOLD_ω is used in place of $\text{UNFOLD}^0_{\gamma_\omega}$. Intuitively, relation UNFOLD^n_ω is used to simulate a one-step move over WIDL-graph γ_ω. The string $h \in \Sigma^n$ is called the n-history, and is needed to record the last n labels seen in order to arrive at the associated $c \in \hat{V}$ string. The string $s \in \mathbf{N}^*$ is called a $\|$-stack, and is needed to record the branches of the $\|$ operators traversed in order to arrive at the associated $c \in \hat{V}$ string.

A one-step move X over a single edge labeled from $\Sigma \cup \{\varepsilon\}$ (condition 2.1) does not modify the $\|$-stack, while appending X to the tail of the current n-history (or simply preserving the n-history if X is ϵ). Upon reaching a node with k out-going $\vdash_{\delta_0}$ labels (condition 2.2), a k-step move is performed simultaneously over all k edges, and consequently all k sub-graphs will be traversed in parallel; the n-history does not change, but the $\|$-stack is appended with the $\langle_{\delta_0}\rangle_{\delta_0}$ symbols. If symbol $\rangle_{\delta_0}$ is the last symbol in the $\|$-stack, a one-step move X over a single edge inside the i-th sub-graph (condition 2.3) replaces symbol $\rangle_{\delta_0}$ with i in the $\|$-stack, while appending X to the tail of the current n-history (or simply preserving

the n-history if X is ϵ). If this single edge is labeled $\dashv^i_{\delta_0}$, the $\|$-stack is appended with symbol $\rangle_{\delta_0}$ (condition 2.4), which indicates that the graph corresponding to argument i has been completely traversed. On the other hand, if the the one-step move over a single edge is along a different argument j of the last $\|$ operator and the last symbol in the $\|$-stack is not symbol $\rangle_{\delta_0}$ (condition 2.5), the current prefix of indexes is replaced with 0 in the $\|$-stack (and thereby indicate an argument-shuffled string), while appending X to the tail of the current n-history (or simply preserving the n-history if X is ϵ). Finally, upon reaching a node with k in-going $\dashv_{\delta_0}$ labels (condition 2.6), a k-step move is again performed simultaneously over all k edges, and the parallel traversal of all k sub-graphs is finished; the n-history does not change, but the $\|$-stack is modified by deleting the permutation sequence between the $\langle_{\delta_0}$ and $\rangle_{\delta_0}$ symbols (in the case of an argument-permutation string), or the 0 symbol (in the case of an argument-shuffle string).

I define now the notion of n-cut for a WIDL-graph γ_ω, based on relation UNFOLD^n_ω. The set $n\text{-}\mathsf{cut}(\omega)$ plays an important role in defining the semantics of WIDL-graphs, as well as in establishing the connection between the WIDL formalism and the formalism of probabilistic finite-state automata.

Definition 5 *Let Σ be a finite alphabet, ω be a WIDL-expression over Σ and Δ, and $\gamma_\omega = (V, E, v_s, v_e, \lambda, r)$ be the associated WIDL-graph. Let n be a natural number, and let Σ^n denote the set of strings over alphabet Σ of length at most n. Let $\mathbf{S} = (\mathbf{N} \cup \{\langle_\delta, \rangle_\delta | \delta \in \Delta\})^*$ denote the set of strings over the set of natural numbers and the set of symbols $\langle_\delta, \rangle_\delta$. The set of all cuts of γ_ω of history length n, written $n\text{-}\mathsf{cut}(\omega)$, is the smallest subset of $\hat{V} \times \Sigma^n \times \mathbf{S}$ satisfying the following two conditions:*

1. *$(v_s, \epsilon, \epsilon)$ belongs to $n\text{-}\mathsf{cut}(\omega)$*

2. *if* $C = (c, h, s) \in n\text{-}\mathsf{cut}(\omega)$, *and* $(C, X, C') \in \text{UNFOLD}^n_\omega$, C' *belongs to* $n\text{-}\mathsf{cut}(\omega)$;

Henceforth, the notation $\mathsf{cut}(\omega)$ is used in place of $0\text{-}\mathsf{cut}(\omega)$. Intuitively, a cut $(v_1 \dots v_k, h, s) \in n\text{-}\mathsf{cut}(\omega)$ corresponds to the "parallel" evaluation of some of the subexpressions of ω, where the $v_i, 1 \leq i \leq k$ refer to these subexpressions, h is the so-called n-history, and s is the $\|$-stack.

2.2.3 The Formal Semantics of WIDL-graphs

For a WIDL-expression ω, let $\gamma_\omega = (V, E, v_s, v_e, \lambda, r)$ be the associated WIDL-graph. Throughout this section, I will use the definitions for UNFOLD^n_ω and $n\text{-}\mathsf{cut}(\omega)$ with $n = 0$, that is, with the n-history always ϵ. Therefore, I will leave out the n-history altogether from the description of UNFOLD_ω.

Let $C', C'' \in \mathsf{cut}(\omega)$ and $w \in \Sigma^*$. I write $w \in L(C', C'')$ if there exists $q \geq |w|, X_i \in \Sigma \cup \{\varepsilon\}, 1 \leq i \leq q$, and $C_i \in \mathsf{cut}(\omega), 0 \leq i \leq q$, such that $X_1 \dots X_q = w, C_0 = C', C_q = C''$ and $(C_{i-1}, X_i, C_i) \in \text{UNFOLD}_\omega$ for $1 \leq i \leq q$. I also define $L(C, C) = \{\varepsilon\}$. This definition is similar with the one for IDL-graphs (Nederhof & Satta 2004a), and, together with Lemma 1, guarantees that $L(v_s, v_e) = \mathsf{dom}(\sigma_{\text{widl}}(\omega))$, that is, the domain of the probability distribution generated by the WIDL-expression ω is the same as the set of strings obtained by traversing the WIDL-graph γ_ω, starting from cut $(v_s, \epsilon, \epsilon)$ and ending in cut $(v_e, \epsilon, \epsilon)$.

The following definition provides the semantics of WIDL-graphs in terms of probability distributions over finite strings.

Definition 6 *Let* Σ *be a finite alphabet,* Δ *be another, disjoint, countably infinite alphabet,* ω *be a WIDL-expression over* Σ *and* Δ, *and* $\gamma_\omega = (V, E, v_s, v_e, \lambda, r)$ *be the associated*

WIDL-graph. Function $\sigma_{\text{widl}}(\gamma_\omega) : \text{Dom}_{\gamma_\omega} \rightarrow [0,1]$ *is the probability distribution associated with WIDL-graph* γ_ω*, where* $\text{Dom}_{\gamma_\omega} = L(v_s, v_e)$*, and the probability values are defined using a function* $\sigma_\Sigma(\gamma_\omega) : \Sigma \cup \{\varepsilon\} \rightarrow [0,1]$*, which assigns probability values to* γ_ω *edge labels as follows:*

1. *if* $(c[v], s, X, c[v := v'], s) \in \text{UNFOLD}_\omega$*, and* $\lambda((v, v')) = X \in \Sigma \cup \{\varepsilon\}$*, and* $(v_0, v_1), \ldots, (v_n, v) \in E$ *is minimal such that* $\lambda((v_0, v_1)) = (\overset{k}{\delta_0}$*, then*

$$\sigma_\Sigma(\gamma_\omega)(X) = \delta_0(k) \tag{2.7}$$

2. *if* $(c[v], s, X_i, c[v := v'], s') \in \text{UNFOLD}_\omega$*, and* $\lambda((v, v')) = X_i \in \Sigma \cup \{\varepsilon\}$*, and* $s = s_1\langle_{\delta_0} x$*,* $s' = s_1 y$*, with* δ_0 *defined, for* n *indexes, explicitly for a permutation set* $Perm$*, and assigning* p_o *probability mass to permutations not in* $Perm$ *using function* f*, and* p_s *probability mass to shuffles; if* $N_o = n! - |Perm|$*, and* $N_s = |\text{UNFOLD}_\omega| - |\text{UNFOLD}_{\omega \| \delta_0 \times}|$*, then*

   (a) *if* $s = s_1\langle_{\delta_0}\rangle_{\delta_0}, s' = s_1\langle_{\delta_0} k$*, then*

$$\begin{aligned}
\sigma_\Sigma(\gamma_\omega)(X_i) &= \Sigma_{kx \in Perm}\delta_0(kx) + \Sigma_{kx \notin Perm}\frac{1}{N_o}p_o + \frac{1}{N_s}p_s \ \text{ if } f = \text{uniform} \qquad (2.8)\\
\sigma_\Sigma(\gamma_\omega)(X_i) &= \Sigma_{kx \in Perm}\delta_0(kx) + \Sigma_{kx \notin Perm}f(kx)p_o + \frac{1}{N_s}p_s \ \text{ if } f = \text{exp_mov_pen}
\end{aligned}$$

   (b) *if* $s = s_1\langle_{\delta_0} s_2\rangle_{\delta_0}, s' = s_1\langle_{\delta_0} s_2 k$*, then*

$$\begin{aligned}
\sigma_\Sigma(\gamma_\omega)(X_i) &= \frac{\Sigma_{s_2 kx \in Perm}\delta_0(s_2 kx) + \Sigma_{s_2 kx \notin Perm}\frac{1}{N_o}p_o + \frac{1}{N_s}p_s}{\Sigma_{s_2 y \in Perm}\delta_0(s_2 y) + \Sigma_{s_2 y \notin Perm}\frac{1}{N_o}p_o + \frac{1}{N_s}p_s} \ \text{ if } f = \text{uniform} \qquad (2.9)\\
\sigma_\Sigma(\gamma_\omega)(X_i) &= \frac{\Sigma_{s_2 kx \in Perm}\delta_0(s_2 kx) + \Sigma_{s_2 kx \notin Perm}f(s_2 kx)p_o + \frac{1}{N_s}p_s}{\Sigma_{s_2 y \in Perm}\delta_0(s_2 y) + \Sigma_{s_2 y \notin Perm}f(s_2 y)p_o + \frac{1}{N_s}p_s} \ \text{ if } f = \text{exp_mov_pen}
\end{aligned}$$

(c) if $s = s_1\langle_{\delta_0} s_2 k\rangle_{\delta_0}, s' = s_1$, *then*

$$\sigma_\Sigma(\gamma_\omega)(X_i) = \frac{\Sigma_{s_2 kx \in Perm}\delta_0(s_2 kx) + \Sigma_{s_2 kx \notin Perm}\frac{1}{N_o}p_o}{\Sigma_{s_2 y \in Perm}\delta_0(s_2 y) + \Sigma_{s_2 y \notin Perm}\frac{1}{N_o}p_o + \frac{1}{N_s}p_s} \text{ if } f = \mathsf{uniform} \quad (2.10)$$

$$\sigma_\Sigma(\gamma_\omega)(X_i) = \frac{\Sigma_{s_2 kx \in Perm}\delta_0(s_2 kx) + \Sigma_{s_2 kx \notin Perm}f(s_2 kx)p_o}{\Sigma_{s_2 y \in Perm}\delta_0(s_2 y) + \Sigma_{s_2 y \notin Perm}f(s_2 y)p_o + \frac{1}{N_s}p_s} \text{ if } f = \mathsf{exp\_mov\_pen}$$

(d) if $s = s_1\langle_{\delta_0} s_2, s' = s_1\langle_{\delta_0} 0$, *then*

$$\sigma_\Sigma(\gamma_\omega)(X_i) = \frac{\frac{1}{N_s}p_s}{\Sigma_{s_2 y \in Perm}\delta_0(s_2 y) + \Sigma_{s_2 y \notin Perm}\frac{1}{N_o}p_o + \frac{1}{N_s}p_s} \text{ if } f = \mathsf{uniform} \quad (2.11)$$

$$\sigma_\Sigma(\gamma_\omega)(X_i) = \frac{\frac{1}{N_s}p_s}{\Sigma_{s_2 y \in Perm}\delta_0(s_2 y) + \Sigma_{s_2 y \notin Perm}f(s_2 y)p_o + \frac{1}{N_s}p_s} \text{ if } f = \mathsf{exp\_mov\_pen}$$

3. for any other $(c[v], s, X, c[v := v'], s) \in \text{UNFOLD}_\omega$, $\sigma_\Sigma(\gamma_\omega)(X) = 1$;

The probability of a string $w \in L(c', c''), w = X_1 \ldots X_q$, $(c_{i-1}, X_i, c_i) \in \text{UNFOLD}_\omega$, $c_i \in \mathtt{cut}(\omega)$ *for* $1 \leq i \leq q$, *is defined as* $\sigma_{\text{widl}}(\gamma_\omega)(w) = \sigma_\Sigma(\gamma_\omega)(X_1) \cdot \ldots \cdot \sigma_\Sigma(\gamma_\omega)(X_q)$.

Equation 2.7 covers the situation when a label X appears along a branch k of a sub-graph corresponding to a $\vee_{\delta_0}$-headed expression, and the probability of this label is given by $\delta_0(k)$.

For a sub-graph corresponding to a $\|_{\delta_0}$-headed expression, the explicitly specified permutations are denoted by $Perm$, and therefore the number of implicitly specified permutations of n indexes, N_o, is $n! - |Perm|$. Also, by considering the expression $\omega_{\|_{\delta_0}\times}$ (in which the arguments of the $\|_{\delta_0}$ operator have been locked), I compute the total number of shuffles as the total number of relations (paths) in UNFOLD_ω minus the total number of relations (paths) in $\text{UNFOLD}_{\omega_{\|_{\delta_0}\times}}$.

Using these notations, Equation 2.8 provides the probability assigned to a first label X_i considered inside a $\|$-headed graph, as the sum between the probability mass of all the

explicitly specified permutations starting with index k, the probability mass of all the remaining permutations (computed using function f), and the probability to produce a shuffled string. Equation 2.9 covers the situation when a label X_i along some branch k is considered, and provides the probability for X_i by dividing the probability mass of the permutations consistent with prefix $s_2 k$ plus the probability mass of a shuffled string to the probability mass of the permutations consistent with prefix s_2 plus the probability mass of a shuffled string.

Equation 2.10 covers the situation when a label X_i finishes the traversal of the $||$-headed graph with a permutation order, and provides the probability for X_i by dividing the probability mass of the permutations consistent with prefix $s_2 k$ to the probability mass of the permutations consistent with prefix s_2 plus the probability mass of a shuffled string. Finally, Equation 2.11 takes care of the situation when considering label X_i in a traversal creates a shuffled string, and provides the probability for X_i by dividing the probability mass of a shuffled string to the probability mass of the permutations consistent with the previous prefix s_2 plus the probability mass of a shuffled string. In any other cases, the probability of a label X in an UNFOLD_ω element is 1.

2.2.4 The Equivalence between WIDL-expressions and WIDL-graphs

In this section, I formulate and prove a theorem concerning the equivalence, at the semantic level, between WIDL-expressions and WIDL-graphs. Although I have provided a mapping at syntactic level between the two representations (Definition 3), which ensures that representation properties transfer between WIDL-expressions and WIDL-graphs, it is the following result that ensures that a WIDL-expression and the WIDL-graph in which it maps have the same interpretation.

Theorem 3 *Let Σ be a finite alphabet, ω be a WIDL-expression over Σ and Δ, and $\gamma_\omega = (V, E, v_s, v_e, \lambda, r)$ be the associated WIDL-graph. Then $\sigma_{\text{widl}}(\omega) = \sigma_{\text{widl}}(\gamma_\omega)$.*

Proof: As $L(v_s, v_e) = \mathsf{dom}(\sigma_{\text{widl}}(\omega))$, from Definition 6 it immediately follows that $\mathsf{dom}(\sigma_{\text{widl}}(\gamma_\omega)) = \mathsf{dom}(\sigma_{\text{widl}}(\omega))$. It follows that one only has to show that, for any w in $\mathsf{dom}(\sigma_{\text{widl}}(\omega))$, $\sigma_{\text{widl}}(\omega)(w) = \sigma_{\text{widl}}(\gamma_\omega)(w)$. The proof goes by structural induction on the structure of ω. In what follows, I will treat only the most interesting cases, involving the $\vee$ and $\|$ operators.

1. $\omega = \vee_{\delta_0}(\omega_1, \ldots, \omega_n), \delta_0 = \{1 \rightarrow p_1, \ldots, n \rightarrow p_n\}$, and ω_i a WIDL-expression, $1 \leq i \leq n$. For any $w \in \mathsf{dom}(\sigma_{\text{widl}}(\omega))$, from Definition 2 there exists $k, 1 \leq k \leq n$, such that $w = w_k$, $w_k \in \mathsf{dom}(\sigma_{\text{widl}}(\omega_k))$, and $\sigma_{\text{widl}}(\omega)(w) = \delta_0(k) \cdot \sigma_{\text{widl}}(\omega)(w_k)$. From Definitions 3 and 4, it follows that

   $\text{UNFOLD}_{\gamma_\omega} = \cup_{i=1}^{n} \text{UNFOLD}_{\gamma_{\omega_i}} \cup \{(v_s, \epsilon, \varepsilon, v_{i,s}, \epsilon), (v_{i,e}, \epsilon, \varepsilon, v_e, \epsilon) | 1 \leq i \leq n\}$,

   for which $\lambda(v_s, v_{i,s}) = (_{\delta_0}^{i}$ and $\lambda(v_{i,e}, v_e) =)_{\delta_0}^{i}$. Then $w \in L(v_s, v_e)$, as a yield of $(v_s, \epsilon, \varepsilon_1, v_{k,s}, \epsilon), \text{UNFOLD}_{\omega_k}, (v_{k,e}, \epsilon, \varepsilon_2, v_e, \epsilon)$. Following Definition 6, $\sigma_{\text{widl}}(\gamma_\omega)(w) = \sigma_{\text{widl}}(\gamma_\omega)(\varepsilon_1) \cdot \sigma_{\text{widl}}(\gamma_\omega)(w_k) \cdot \sigma_{\text{widl}}(\gamma_\omega)(\varepsilon_2)$. As $\lambda(v_s, v_{k,s}) = (_{\delta_0}^{k}$, Equation 2.7 one gets $\sigma_{\text{widl}}(\gamma_\omega)(\varepsilon_1) = \delta_0(k)$. As $\lambda(v_{k,e}, v_e) =)_{\delta_0}^{k}$, from the default rule one gets $\sigma_{\text{widl}}(\gamma_\omega)(\varepsilon_2) = 1$. It follows that $\sigma_{\text{widl}}(\gamma_\omega)(w) = \delta_0(k) \cdot \sigma_{\text{widl}}(\gamma_\omega)(w_k) \cdot 1$.

   The induction hypothesis is $\sigma_{\text{widl}}(\gamma_\omega)(w_k) = \sigma_{\text{widl}}(\omega)(w_k)$, and therefore it follows that $\sigma_{\text{widl}}(\gamma_\omega)(w) = \delta_0(k) \cdot \sigma_{\text{widl}}(\omega)(w_k) = \sigma_{\text{widl}}(\omega)(w)$.

2. $\omega = \|_{\delta_0}(\omega_1, \ldots, \omega_n), \delta_0 = \{perm_e \rightarrow p_e, \mathsf{other\ perm} \xrightarrow{f} p_o, \mathsf{shuffles} \xrightarrow{\mathsf{uniform}} p_s\}$, $f \in \{\mathsf{uniform}, \mathsf{exp\_mov\_pen}\}$, and ω_i a WIDL-expression, $1 \leq i \leq n$. (The cases when there are more than one – or none – explicit permutation can be treated in the same manner.) For any $w \in \mathsf{dom}(\sigma_{\text{widl}}(\omega))$, it follows from Definition 2 that w can

be written as either permutation $i_1 \ldots i_n = perm_e$ of some $w_i \in \mathsf{dom}(\sigma_{\text{widl}}(\omega_i))$, $1 \leq i \leq n$, or some other permutation, or no permutation (for shuffled strings). I only expand here the first case, as the other two cases use similar arguments. If $w = w_{i_1} \ldots w_{i_n}$, then $\sigma_{\text{widl}}(\omega)(w) = p_e \cdot \sigma_{\text{widl}}(\omega)(w_{i_1}) \cdot \ldots \cdot \sigma_{\text{widl}}(\omega)(w_{i_n})$. From Definition 3 and Definition 4, it follows that

(a) $(v_s, \epsilon, \varepsilon, v_{1,s}v_{2,s} \ldots v_{n,s}, \langle_{\delta_0}\rangle_{\delta_0}) \in \text{UNFOLD}_{\gamma_\omega}$

(b) $(c[v_{i_1,s}], \langle_{\delta_0}\rangle_{\delta_0}, X^1_{i_1}, c[v^1_{i_1}], \langle_{\delta_0} i_1), \ldots, (c[v^{m_1-1}_{i_1}], \langle_{\delta_0} i_1, X^{m_1}_{i_1}, c[v_{i_1,e}], \langle_{\delta_0} i_1)$, $(c[v_{i_1,e}], \langle_{\delta_0} i_1, \varepsilon, c[v_{i_1,e}], \langle_{\delta_0} i_1\rangle_{\delta_0}) \in \text{UNFOLD}_{\gamma_\omega}$

(c) $(c[v_{i_2,s}], \langle_{\delta_0} i_1\rangle_{\delta_0}, X^1_{i_2}, c[v^1_{i_2}], \langle_{\delta_0} i_1 i_2), \ldots, (c[v^{m_2-1}_{i_2}], \langle_{\delta_0} i_1 i_2, X^{m_2}_{i_2}, c[v_{i_2,e}], \langle_{\delta_0} i_1 i_2)$, $(c[v_{i_2,e}], \langle_{\delta_0} i_1 i_2, \varepsilon, c[v_{i_2,e}], \langle_{\delta_0} i_1 i_2\rangle_{\delta_0}) \in \text{UNFOLD}_{\gamma_\omega}, \ \ldots$

(d) $(v_{i_1,e}v_{i_2,e} \ldots v_{i_n,e}, \langle_{\delta_0} i_1 i_2 \ldots i_n, \varepsilon, v_{i_1,e}v_{i_2,e} \ldots v_{i_n,e}, \langle_{\delta_0} i_1 i_2 \ldots i_n\rangle_{\delta_0}) \in \text{UNFOLD}_{\gamma_\omega}$

(e) $(v_{i_1,e}v_{i_2,e} \ldots v_{i_n,e}, \langle_{\delta_0} i_1 i_2 \ldots i_n\rangle_{\delta_0}, \varepsilon, v_e, \epsilon) \in \text{UNFOLD}_{\gamma_\omega}$

for which $w_{i_1} = X^1_{i_1} \ldots X^{m_1}_{i_1}$, $w_{i_2} = X^1_{i_2} \ldots X^{m_2}_{i_2}$, etc., and $w \in L(v_s, v_e)$ is a yield of the $\text{UNFOLD}_{\gamma_\omega}$ elements enumerated at (a)-(e). Following Definition 6, one has $\sigma_{\text{widl}}(\gamma_\omega)(w) = \sigma_{\text{widl}}(\gamma_\omega)(X^1_{i_1}) \cdot \ldots \cdot \sigma_{\text{widl}}(\gamma_\omega)(X^{m_1}_{i_1}) \cdot \sigma_{\text{widl}}(\gamma_\omega)(X^1_{i_2}) \cdot \ldots$ $\cdot\sigma_{\text{widl}}(\gamma_\omega)(X^{m_n}_{i_n})$. From Equations 2.8-2.10, it follows that $\sigma_{\text{widl}}(\gamma_\omega)(X^1_{i_1}) \cdot \sigma_{\text{widl}}(\gamma_\omega)(X^1_{i_2}) \cdot$ $\ldots \cdot \sigma_{\text{widl}}(\gamma_\omega)(X^1_{i_n}) = p_e$ (the denominator of each term cancels out with the numerator of the precedent term, except for the last numerator, which is p_e), while the remaining terms are computed for each $\text{UNFOLD}_{\gamma_{\omega_i}}$ individually, and yield $\sigma_{\text{widl}}(\gamma_\omega)(w_{i_1}) \cdot$ $\ldots \cdot \sigma_{\text{widl}}(\gamma_\omega)(w_{i_n})$. From the induction hypothesis, $\sigma_{\text{widl}}(\gamma_\omega)(w_i) = \sigma_{\text{widl}}(\omega)(w_i)$, $1 \leq i \leq n$, and therefore $\sigma_{\text{widl}}(\gamma_\omega)(w) = p_e \cdot \sigma_{\text{widl}}(\omega)(w_{i_1}) \cdot \ldots \cdot \sigma_{\text{widl}}(\omega)(w_{i_n}) =$ $\sigma_{\text{widl}}(\omega)(w)$.

2.3 WIDL-graphs and Probabilistic Finite-State Acceptors

Probabilistic finite-state acceptors (pFSA) are a well-known formalism for representing probability distributions (Mohri, Pereira, & Riley 2002; Knight & Graehl 1998). In this section, I discuss the correspondence between the formalism of WIDL-expressions and the pFSA formalism. This correspondence is based on the relation UNFOLD_ω and the set $\mathsf{cut}(\omega)$ defined for a WIDL-graph γ_ω. In what follows, I will present informally the relation UNFOLD_ω, in the context of this correspondence between formalisms. For a formal description of this relation, see the formal fragments of Section 2.2.

I will use Figure 2.3 as a running example for the correspondence between WIDL-graphs and pFSA. The WIDL-graph in Figure 2.3(a) is the same as the WIDL-graph in Figure 2.2, corresponding to the WIDL-expression in Figure 2.1. In general, for a WIDL-expression ω over a finite alphabet Σ, the mechanism for mapping the WIDL-graph γ_ω into a pFSA A_ω creates a state s in A_ω for each set of WIDL-graph vertices that can be reached simultaneously when traversing the graph, while recording, in what I call a $\|$-stack (interleave stack), the order in which $\vdash^i_\delta$,$\dashv^i_\delta$–bordered sub-graphs are traversed. Consider Figure 2.3(b), in which state $[v_0 v_9 v_{23}, \langle_{\delta_1} 3\ 2]$ (at the bottom) corresponds to reaching vertices v_0, v_9, and v_{23} (see the WIDL-graph in Figure 2.3(a)), by first reaching vertex v_{23} (inside the $\vdash^3_{\delta_1}$, $\dashv^3_{\delta_1}$–bordered sub-graph), and then reaching vertex v_9 (inside the $\vdash^2_{\delta_1}$, $\dashv^2_{\delta_1}$–bordered sub-graph). A transition labeled $a \in \Sigma$ between two A_ω states s_1 and s_2 in A_ω exists if there exists a vertex v_j in the description of s_1 and a vertex v_k in the description of s_2 such that there exists a path in γ_ω between v_j and v_k, and a is the only Σ-labeled transitions in this path.

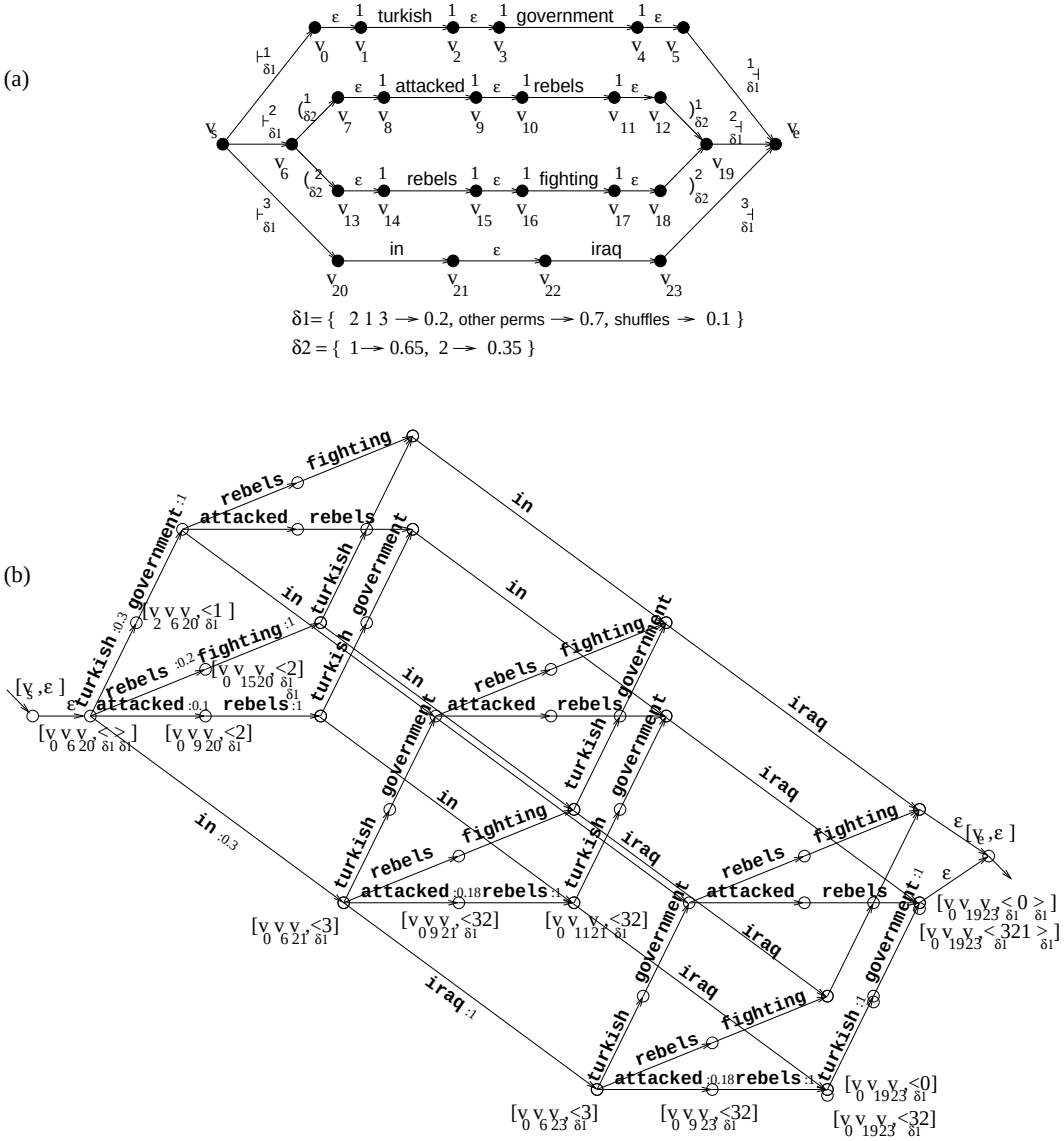

Figure 2.3: The WIDL-graph corresponding to the WIDL-expression in Figure 2.1 is shown in (a). The probabilistic finite-state acceptor (pFSA) that corresponds to the WIDL-graph is shown in (b).

For example, transition $[v_0 v_9 v_{23}, \langle_{\delta_1} 3\, 2] \overset{\textsf{rebels}}{\rightarrow} [v_0 v_{19} v_{23}, \langle_{\delta_1} 3\, 2]$ (Figure 2.3(b)) results from unfolding the path $v_9 \overset{\varepsilon}{\rightarrow} v_{10} \overset{\textsf{rebels}}{\rightarrow} v_{11} \overset{\varepsilon}{\rightarrow} v_{12} \overset{)^1_{\delta_2}}{\rightarrow} v_{19}$ (Figure 2.3(a)). A transition labeled ε between two A_ω states s_1 and s_2 in A_ω exists if there exists a vertex v_j in the description of s_1 and vertices $v_k^1, \ldots, v_k^n$ in the description of s_2, such that $v_j \overset{\vdash^i_\delta}{\rightarrow} v_k^i \in \gamma_\omega$, $1 \leq i \leq n$ (see transition $[v_s, \epsilon] \overset{\varepsilon}{\rightarrow} [v_0 v_6 v_{20}, \langle_{\delta_1}]$), or if there exists vertices $v_j^1, \ldots, v_j^n$ in the description of s_1 and vertex v_k in the description of s_2, such that $v_j^i \overset{\dashv^i_\delta}{\rightarrow} v_k \in \gamma_\omega$, $1 \leq i \leq n$. These ε-transitions are responsible for adding and removing, respectively, the $\langle_\delta$, $\rangle_\delta$ symbols in the $\|$-stack. The probabilities associated with A_ω transitions are computed using the information present in the description of A_ω states and the operator distribution functions, according to the equations used in Definition 6.

At a formal level, one can use the definitions for the relation UNFOLD^n_ω and set n-$\textsf{cut}(\omega)$ (Section 2.2, Definitions 4 and 5, respectively) to make the connection between the WIDL representation and the pFSA representation, as follows. For a WIDL-expression ω, the finite set of states of its corresponding pFSA is the set $\textsf{cut}(\omega)$; the transitions between the pFSA states are given by the relation UNFOLD_ω; the initial state of the pFSA is the cut $[v_s, \epsilon]$, and the final state is the cut $[v_e, \epsilon]$. If one is interested in having pFSA whose probabilistic transitions reflect the probability distributions of n-gram language models (Mohri, Pereira, & Riley 2002; Knight & Graehl 1998), then the finite set of states is given by the set $(n-1)$-$\textsf{cut}(\omega)$, the transitions are given by the relation $\text{UNFOLD}^{n-1}_\omega$, the initial state is the cut $[v_s, \epsilon]$, and the final states are the cuts $[v_e, h, \epsilon]$, where h is a $(n-1)$-history string.

Let us emphasize now the differences between the WIDL representation and the pFSA representation. While the WIDL representation is linear in the number of words used for generation (Theorem 2), the pFSA representation may grow exponentially in the number of words used. Two different factors are responsible, both related to the $\|_\delta$ operators. First, a pFSA representation must explicitly enumerate all possible interleavings, so that an n-ary $\|$ operator that requires only $2n$ edges in a WIDL-graph becomes a n-dimensional cube (2^n edges) in a pFSA. Second, a pFSA representation needs to explicitly record the order of traversal of the arguments of the $\|$ operators, in order to correctly compute transition probabilities. One should observe here that the cubes that form the pFSA shown in Figure 2.3(b) are drawn as such only to facilitate visualization. In reality, different unfolding orders do not converge into the same state, as state $[v_0 v_{19} v_{23}, \langle_{\delta_1} 3\, 2]$ records argument order 3 and 2 for $\|_{\delta_1}$, whereas state $[v_0 v_{19} v_{23}, \langle_{\delta_1} 0]$, records a shuffled order for $\|_{\delta_1}$ (see the bottom-right part of Figure 2.3(b)). An n-ary $\|$ operator, therefore, produces an order of $n!$ different configurations of argument orderings, which need to be recorded explicitly in the pFSA representation.

2.4 Characteristics of WIDL-expressions

In this section, I make use of the formal notions UNFOLD_ω and $\mathsf{cut}(\omega)$ (Section 2.1, Definitions 4 and 5) to define functions that characterize some of the properties of WIDL-expressions.

Definition 7 *Let ω be a WIDL-expression and $\gamma_\omega = (V, E, v_s, v_e, \lambda, r)$ be the associated WIDL-graph. If $\mathsf{cut}(\omega) \subseteq \hat{V} \times \mathbf{S}$ is the set of all cuts of γ_ω, then function pos is defined as:*

$$\mathsf{pos} : \mathsf{cut}(\omega) \rightarrow \hat{V} \quad \mathsf{pos}([C, s]) = C$$

Function pos is a projection function for the set of all cuts of γ_ω. It deletes all the information regarding the $\|$-stack, and keeps only the position in the "parallel" evaluation of ω corresponding to the cut considered.

The next definition introduces the notion of the width of a WIDL-expression. This notion directly affects the efficiency of algorithms that use the WIDL formalism, as described in Chapter 3.

Definition 8 *Let ω be a WIDL-expression and γ_ω be the associated WIDL-graph. Function* width *is defined by:*

$$\mathsf{width}(\omega) = \mathsf{max}_{c \in \mathsf{cut}(\omega)} |\mathsf{pos}(c)|$$

Function width returns the maximum number of vertices from γ_ω present in a cut. For a given WIDL-expression ω, $\mathsf{width}(\omega)$ is one of the quantities used to measure the complexity of ω.

Next definition provides a way to characterize a WIDL-expression ω via the length of the shortest string in $\mathsf{dom}(\sigma_{\text{widl}}(\omega))$. This notion will be used to compute admissible costs for the set of strings defined by $\mathsf{dom}(\sigma_{\text{widl}}(\omega))$, as described in Chapter 8.

Definition 9 *Let ω be a WIDL-expression over Σ and Δ. Function* $\mathsf{shortest}(\omega)$ *is defined as follows:*

1. $\mathsf{shortest}(a) = 1, a \in \Sigma, \mathsf{shortest}(\mathcal{E}) = 0$;

2. $\mathsf{shortest}(\vee_\delta(\omega_1, \ldots, \omega_n)) = \mathsf{min}_i\mathsf{shortest}(\omega_i)$;

3. $\mathsf{shortest}(\|_\delta(\omega_1, \ldots, \omega_n)) = \Sigma_i\mathsf{shortest}(\omega_i)$;

4. $\mathsf{shortest}(\omega_1 \cdot \omega_2) = \mathsf{shortest}(\omega_1) + \mathsf{shortest}(\omega_2)$;

5. $\mathsf{shortest}(\times(\omega_1)) = \mathsf{shortest}(\omega_1)$;

For expressions headed by operators other than $\vee$, shortest is the sum of the shortest values for the arguments. For expressions headed by operator $\vee$, shortest considers the minimum shortest value of the arguments. For example, for the WIDL-expression in Figure 2.1 (Section 2.1), $\mathsf{shortest}(\omega) = 2 + \min(2, 2) + 2 = 6$.

The next definition concerns a function that allows us to characterize WIDL-expressions in terms of the number of operators occurring within these expressions.

Definition 10 *Let ω be a WIDL-expression over Σ and Δ. Function $\mathsf{nop}(\omega)$ is defined as follows:*

1. $\mathsf{nop}(a) = 0, a \in \Sigma \cup \{\mathcal{E}\}$;

2. $\mathsf{nop}(\vee_\delta(\omega_1, \ldots, \omega_n)) = 1 + \mathsf{max}_i\mathsf{nop}(\omega_i)$;

3. $\mathsf{nop}(\|_\delta(\omega_1, \ldots, \omega_n)) = 1 + \Sigma_i\mathsf{nop}(\omega_i)$;

4. $\mathsf{nop}(\omega_1 \cdot \omega_2) = 1 + \mathsf{nop}(\omega_1) + \mathsf{nop}(\omega_2)$;

5. $\mathsf{nop}(\times(\omega_1)) = 1 + \mathsf{nop}(\omega_1)$;

Function nop measures the complexity of a WIDL-expression by the number of operator occurrences within that expression. For expressions headed by operators other than $\vee$, nop is computed adding 1 (for the head operators) to the sum of the nop values of the arguments. For expressions headed by operator $\vee$, nop is computed adding 1 (for the $\vee$ operator) to the maximum nop value of the arguments. For example, for the WIDL-expression in Figure 2.1 (Section 2.1), $\mathsf{nop}(\omega) = 1 + (2 + (1 + (\mathsf{max}(1 + 1, 1 + 1))) + 1) = 1 + (2 + 3 + 1) = 7$.

2.5 Conclusions

In this chapter, I introduced the formalism of WIDL-expressions, a representation formalism that I will use to integrate a generic natural language generation system within end-to-end language applications. I presented the syntax and semantics of WIDL-expressions, and also the syntax and semantics of an equivalent representation formalism, called WIDL-graphs. The two representations are equivalent both at the syntactic level and semantic level. At the syntactic level, they share the property of growing linearly in the number of atomic elements. At the semantic level, I have shown that a WIDL-expression and its corresponding WIDL-graph represent the same probability distribution over finite strings.

WIDL-expressions and WIDL-graphs complement each other nicely. The advantage of WIDL-expressions is that they have a simple syntax and a simple semantics (in terms of probability distributions over finite strings), defined recursively over WIDL-expression terms. The advantage of WIDL-graphs is that, while equivalent with WIDL-expressions, they allow formulation of algorithms that process them. In the next chapter, I will provide a family of algorithms

that operate on WIDL-graphs, and implement search strategies that allow one to develop a generic natural language generation system that uses the WIDL formalism at its input representation.

Chapter 3

Algorithms for Intersecting WIDL-expressions with n-Gram Language Models

WIDL-expressions provide the means to rank the encoded realizations using the distribution functions associated with its $\|$ and $\vee$ operators. These distribution functions, however, are necessarily underspecified, to preserve the compactness of the representation (Chapter 2, Theorem 2). To compensate, additional probability distributions, usually in the form of stochastic language models, are intersected with the probability distributions represented by WIDL-expressions. As WIDL-expressions are agnostic to any type of language theory, this mechanism allows to integrate the constraints of particular language theories in the generation process, in a mathematically sound fashion.

In this chapter, I present a novel generation mechanism that intersects WIDL-expressions with n-gram language models. The intersection is realized using a log-linear combination of feature functions that characterize WIDL probability distributions and n-gram language model probability distributions. The generation mechanism implements new algorithms, which cover a

wide spectrum of run-time behaviors (from linear to exponential), depending on the complexity of the input WIDL-expression. I also present theoretical results concerning the correctness (i.e., the realization corresponding to the most probable path under a log-linear combination of the WIDL probability distribution and the given language model probability distribution is found) and the efficiency (i.e., time complexity increases linearly with the complexity of the input WIDL-expression) of these algorithms.

3.1 n-gram Language Models

Language models are mathematical devices used to measure the goodness of some sequence of words $w_1 \ldots w_m$ with respect to what humans consider to be a "correct" sequence in natural language. More precisely, a language model assigns a probability $P(w_1 \ldots w_m)$ to sequences of words $w_1 \ldots w_m$, such that the sum of all sequences of m words, $m \geq 1$, is 1. Language models are good if they assign high probability to correct sequences and low probability to incorrect sequences.

The most popular language models to date are the n-gram language models. For n=2, for instance, a 2-gram (bigram) language model approximates the probability $P(w_1 \ldots w_m)$ as $P(w_1) \cdot P(w_2|w_1) \cdot \ldots P(w_m|w_{m-1})$. That is, it estimates the probability of the sequence as a product of the probability of each word. To estimate the probability of a word, the model remembers the last n-1 words seen, and uses them as its context.

The advantages of n-gram language models include well-studied smoothing techniques (Goodman 2001), scalability to large amounts of training data, and vast amounts of training

instances, as only plain monolingual text is required for training. Moreover, there exist publicly-available n-gram language model toolkits, such as the SRI Language Model Toolkit[1], that can be used to train and employ n-gram language models within applications. As a consequence, n-gram language models are currently used by virtually all state-of-the-art speech-recognition and machine translation systems. For this reason, I concentrate in this dissertation on n-gram language models for the purpose of creating a generic framework for stochastic natural language generation.

3.2 Log-linear Interpolation of Probability Distributions

I interpolate the probability distributions of WIDL-expressions with the probability distributions of n-gram language models using a log-linear framework (Papineni, Roukos, & Ward 1997).

Let us assume a finite set E of strings over a finite alphabet Σ, representing the set of possible realizations. In the log-linear framework, one considers a vector of feature functions $\overline{h} = \langle h_0, h_1, \ldots, h_M \rangle$, and a vector of parameters $\overline{\lambda} = \langle \lambda_0, \lambda_1, \ldots, \lambda_M \rangle$. For any $e \in E$, the interpolated probability $P(e)$ can be written under the log-linear model as in Equation 3.1:

$$P(e) = \frac{\exp[\Sigma_{m=0}^{M} \lambda_m h_m(e)]}{\Sigma_{e'} \exp[\Sigma_{m=0}^{M} \lambda_m h_m(e')]} \tag{3.1}$$

[1] http://www.speech.sri.com/projects/srilm/

One can formulate the search problem of finding the most probable realization e under this model as in Equation 3.2:

$$\arg\max_e P(e) = \arg\max_e \exp[\Sigma_{m=0}^{M} \lambda_m h_m(e)] \tag{3.2}$$

$$\arg\max_e P(e) = \arg\min_e -\Sigma_{m=0}^{M} \lambda_m \log h_m(e)$$

and, therefore, one does not need to be concerned about computing expensive normalization factors.

For a given WIDL-expression ω over Σ, the set E is defined by $\mathsf{dom}(\sigma_{\text{widl}}(\omega))$, and feature function h_0 is taken to be $\sigma_{\text{widl}}(\omega)$. Any language model one wants to employ may be added in Equation 3.2 as a feature function h_i, $i \geq 1$.

3.3 WIDL-graphs and n-Gram Language Models

As described in Chapter 2, Section 2.3, for any WIDL-graph γ_ω there exists an equivalent probabilistic finite-state acceptor A_ω, which is obtained by completely unfolding γ_ω, that is, by computing the relation UNFOLD_ω. Consider, for instance, the following WIDL-expression:

$$\omega = \|_u(\mathsf{prisoners}, \mathsf{were}, \mathsf{released}),\ u = \{\mathsf{perms} \overset{\mathsf{uniform}}{\rightarrow} 1\}, \tag{3.3}$$

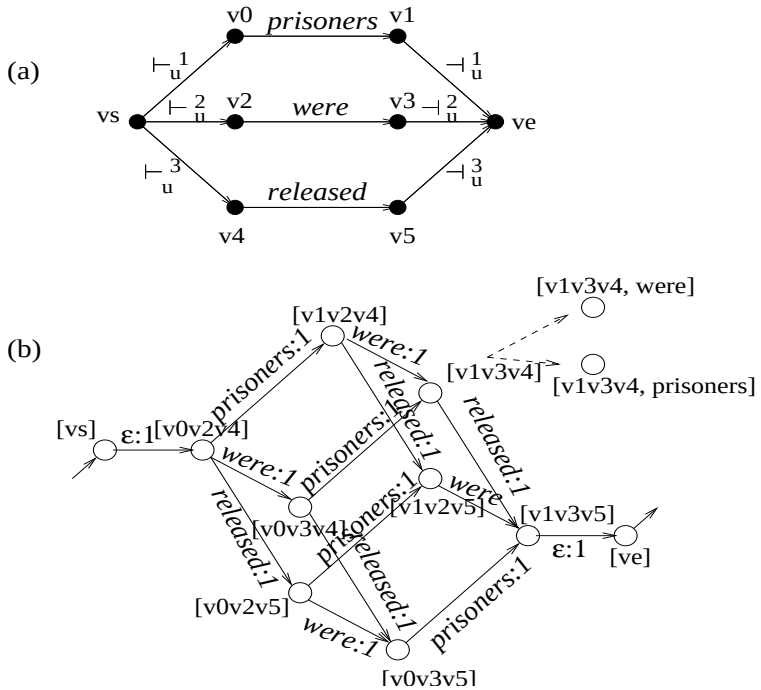

Figure 3.1: The WIDL-graph corresponding to WIDL-expression 3.3 (a), and the weighted finite-state acceptor corresponding to this WIDL-graph (b) (compacted because of the uniform probability).

for which the WIDL-graph γ_ω and its corresponding pFSA A_ω are shown in Figure 3.1 [2].

If one is interested in having a pFSA whose probabilistic transitions reflect the probability distributions of n-gram language models, one needs to split the states of A_ω according to the context information needed by these language models. This corresponds to the information recorded by the n-history strings used to define relation UNFOLD^n_ω and set $n\text{-}\mathtt{cut}(\omega)$ (Chapter 2, Section 2.2). For example, to reflect the probability distribution of a 2-gram language model LM, state $(v_1 v_3 v_4)$ in Figure 3.1 must record the 1-history of the transition last used to reach

[2] One should note that A_ω is actually a compacted version of the pFSA that is obtained following the mapping mechanism described in Section 2.3, as the uniform probability of operator $\|_u$ allows us to losslessly recombine the WIDG cuts that have different $\|$-stack strings.

this state. The result is states $(v_1 v_3 v_4, \textsf{prisoners})$ and $(v_1 v_3 v_4, \textsf{were})$. The transitions leaving these states have the same labels as those leaving the original state $(v_1 v_3 v_4)$, and can now be weighted using a linear combination between the original WIDL probabilities and n-gram language model probabilities, such as $p_{LM}(\cdot|\textsf{prisoners})$ and $p_{LM}(\cdot|\textsf{were})$.

At this point, one already has a naïve algorithm for intersecting WIDL-expressions with n-gram language models. From a WIDL-expression ω, one first computes the relation $\text{UNFOLD}_{\omega}^{n-1}$ and the set $(n-1)$-$\textsf{cut}(\omega)$, using the corresponding WIDL-graph γ_ω. These sets define the transition and state set of the probabilistic finite-state acceptor, henceforth called A_{ω}^{n-1}. On A_{ω}^{n-1}, one can use a single-source shortest-path algorithm for directed acyclic graphs (Cormen *et al.* 2001) to extract the realization corresponding to the most probable path under a log-linear combination of WIDL probability distributions and n-gram language model probability distributions (Equation 3.2). The problem with this algorithm, however, is that a premature unfolding of the WIDL-graph into a probabilistic finite-state acceptor destroys the representation compactness of the WIDL representation (Chapter 2, Section 2.3).

For this reason, I devise algorithms that, although similar in spirit with the single-source shortest-path algorithm for directed acyclic graphs, perform on-the-fly unfolding of the WIDL-graph, with a mechanism to control the unfolding based on the costs of the paths already unfolded. This approach has the advantage that prefixes that are extremely unlikely under the combination of WIDL costs and language model costs may be regarded as not so promising, and parts of the WIDL-expression that contain them may not be unfolded, leading to significant space and time savings.

```
WIDL-NGLM-BFS(γ_ω, LM̄, λ̄)
1  active ← {[v_s^{γ_ω}, ε, ε]}
2  flag ← 1
3  while flag
4       do unfold ← UNFOLD(active, γ_ω, LM̄)
5          EVALUATE(unfold, LM̄, λ̄)
6          if FINAL(unfold, γ_ω)
             then flag ← 0
7          active ← unfold
8  return active
```

Figure 3.2: Pseudo-code for intersecting a WIDL-graph γ_ω with n-gram language models $\overline{LM}$ using incremental unfolding and breadth-first search.

3.4 Generation via Intersection of WIDL-expressions with n-Gram Language Models

Algorithm WIDL-NGLM-BFS The first algorithm I propose is WIDL-NGLM-BFS in Figure 3.2. Algorithm WIDL-NGLM-BFS solves the search problem defined by Equation 3.2 for a WIDL-graph γ_ω (which provides feature function h_0) and M n-gram language models (which provide feature functions $h_1, \ldots, h_M$). Throughout this section, $n-1$ is taken to be the maximum size of the context required by any of the M language models considered, and therefore language models that use different context sizes can be treated uniformly.

The algorithm computes incrementally the relation $\text{UNFOLD}_\omega^{n-1}$ for γ_ω (see Chapter 2, Section 2.2) from a set of active states, called $active$ (line 4). The incrementality comes from adding new elements to $\text{UNFOLD}_\omega^{n-1}$, by unfolding the WIDL-graph γ_ω from left to right, starting with the initial state of A_ω^{n-1}, $[v_s^{\gamma_\omega}, \varepsilon, \varepsilon]$. The set of newly UNFOLDed states is called $unfold$.

```
WIDL-NGLM-A*(γ_ω, LM̄, λ̄)
1  active ← {[v_s^{γ_ω}, ε, ε]}
2  flag ← 1 Q ← ∅
3  while flag
4       do unfold ← UNFOLD(active, γ_ω, LM̄)
5          EVALUATE(unfold, LM̄, λ̄)
6          if FINAL(unfold, γ_ω)
             then flag ← 0
7          for each state in unfold
               do PUSH(Q, state)
8          active ← POP(Q)
9  return active
```

Figure 3.3: Pseudo-code for intersecting a WIDL-graph γ_ω with n-gram language models $\overline{LM}$ using incremental unfolding and A* search.

Using Equation 3.2, the algorithm EVALUATEs the current $P(e)$ costs for the $unfold$ states, by extracting e as the yield of the path from the initial state $[v_s^{\gamma_\omega}, \varepsilon, \varepsilon]$ to each state in $unfold$.

If a final state of A_ω^{n-1} is not yet reached (line 6), the while loop is closed by making the $unfold$ set of states to be the next set of $active$ states. This algorithm is actually a breadth-first search (BFS) for which the A_ω^{n-1} automaton is computed incrementally, by traversing γ_ω from left to right. This algorithm still unfolds the WIDL-graph completely, and therefore suffers from the same drawback as the naïve algorithm described in Section 3.3.

The interesting contribution of algorithm WIDL-NGLM-BFS, however, is the incremental unfolding. If, instead of line 7 in Figure 3.2, I introduce mechanisms to control which $unfold$ states become part of the $active$ set of states in the next unfolding iteration, I obtain a series of more effective algorithms.

Algorithm WIDL-NGLM-A* I arrive at algorithm WIDL-NGLM-A* by modifying line 7 in Figure 3.2, thus obtaining the algorithm in Figure 3.3. I use as control mechanism a priority queue Q, in which the states from $unfold$ are PUSHed, sorted according to a cost associated with each state. EVALUATE uses an admissible heuristic function (Russell & Norvig 1995) to compute future (admissible) costs for the $unfold$ states. The priority queue Q sorts (from min to max) the states according to their total cost (current + admissible). In the next iteration, $active$ is a singleton set containing the state POPed out from the top of Q (the lowest cost state). Using this control mechanism, relation UNFOLD$_\omega^{n-1}$ is computed incrementally, by adding one new state and transition per iteration, in the order of increasing total cost for the newly added states.

In the experiments that I run, I also use a greedy version of this algorithm, called the k-steps version. The k-steps version records the maximum length L of the path from the initial state to all previously considered $active$ states. Instead of performing only one POP at line 8, it POPs from Q until the path from the initial state to the current $active$ state has length greater than or equal to $L-k$. This modification contributes to a significant speed-up of the algorithm, because states that lag behind more than k-steps from the state with maximum path length are simply discarded. Of course, greedily discarding these states may affect the accuracy of the solution found, especially for low k values. However, as k gets larger, the k-steps version approximates better the behavior of the full A* algorithm.

Algorithm WIDL-NGLM-BEAM I arrive at algorithm WIDL-NGLM-BEAM by again modifying line 7 in Figure 3.2, thus obtaining the algorithm in Figure 3.4. I control the unfolding using a value $beam$ and the BEAMSTATES function. The WIDL-NGLM-BEAM algorithm

```
WIDL-NGLM-BEAM(γ_ω, LM̄, λ̄, beam)
1  active ← {[v_s^{γ_ω}, ε, ε]}
2  flag ← 1 B ← ∅
3  while flag
4       do unfold ← UNFOLD(active, γ_ω, LM̄)
5          EVALUATE(unfold, LM̄, λ̄)
6          if FINAL(unfold[0], γ_ω)
             then flag ← 0
7          active ← BEAMSTATES(unfold, beam)
           for each state in active
               do if FINAL(state, γ_ω)
                     then PUSH(B, state)
8  SORT(B)
9  return POP(B)
```

Figure 3.4: Pseudo-code for intersecting a WIDL-graph γ_ω with n-gram language models $\overline{LM}$ using incremental unfolding and beam search.

comes in two flavors: probabilistic beam and histogram beam. In the case when a probabilistic beam is used, function BEAMSTATES selects as $active$ states only the states in $unfold$ reachable with a cost higher or equal to the current minimum cost times $beam$. In the case when a histogram beam is used, function BEAMSTATES selects as $active$ states only the lowest cost $beam$ states.

From the $active$ states, those that are final are recorded in a queue B. The unfolding stops when the lowest-scoring state in the $unfold$ set is also final. At this point, the algorithm sorts the queue B of final states and returns the first state in the queue, i.e., the one with the lowest cost.

Using this control mechanism, relation $\text{UNFOLD}_\omega^{n-1}$ is computed incrementally, by adding at each iteration all the states and transitions that the beam keeps active. The advantage over

the A* version is that the computation of UNFOLD$^{n-1}_{\omega}$ is sped up by adding multiple states and transitions at once. The disadvantage is that states are greedily discarded by the BEAMSTATES function, and are consequently no longer considered for computing the unfolding of γ_ω, which may affect the quality of the path returned.

3.5 Admissible Heuristics for WIDL-expressions Intersected with n-gram Language Models

The WIDL representation is ideally suited for computing accurate admissible heuristics under n-gram language models. These heuristics are needed by the WIDL-NGLM-A* algorithm, and are also employed for pruning by the WIDL-NGLM-BEAM algorithm.

For a given WIDL-graph γ_ω, the corresponding pFSA A^{n-1}_{ω} can be incrementally computed as shown in Section 3.4. At each step in the incremental computation of UNFOLD$^{n-1}_{\omega}$ (when using n-gram language models), one can efficiently extract from γ_ω – without further unfolding – the set[3] of all edge labels that can be used to reach a final state in A^{n-1}_{ω}. This set of labels, denoted FE^{all}_{S}, is an overestimation of the set of future events reachable from S, because the labels under the $\vee$ operators are all considered (whereas, normally, the ones used by different arguments of a $\vee$ operator are mutually exclusive). From FE^{all}_{S} and the $(n-1)$-history string recorded in state S, I obtain a set of label sequences of length $n-1$, denoted FCE_S. This set is an (over)estimated set of the possible future conditioning events for state S, guaranteed to contain the most cost-efficient conditioning events in the future of state S. Using FCE_S,

[3] Actually, these are multisets, as I treat multiply-occurring labels as separate items.

one needs to extract from FE_S^{all} the set of most cost-efficient future events from under each $\vee$ operator. The sets FE_S and FCE_S are used to formulate an admissible heuristic cost for state S under a log-linear combination of M n-gram language models, using Equation 3.4:

$$h_\omega(S) = \Sigma_{e \in FE_S} \Sigma_{m=1}^{M} \lambda_m \min_{ce \in FCE_S} -\log p_{LM_m}(e|ce) \tag{3.4}$$

If $h_\omega^*(S)$ is the true future model cost for state S, Equation 3.4 guarantees that $h_\omega(S) \leq h_\omega^*(S)$ from the way the set of future events FE_S and the set of future conditioning events FCE_S are constructed. One should note that, as it usually happens with admissible heuristics, $h_\omega(S)$ can be made arbitrarily close to $h_\omega^*(S)$, by computing increasingly better approximations FCE_S of FCE_S^*. Such approximations, however, require increasingly advanced unfoldings of the WIDL-graph γ_ω (a complete unfolding of γ_ω for state S gives $FCE_S = FCE_S^*$, and consequently $h_\omega(S) = h_\omega^*(S)$). It follows that arbitrarily accurate admissible heuristics exist for WIDL-expressions, but computing them on-the-fly requires finding a balance between the time and space requirements for computing better heuristics and the speed-up obtained by using them in the search algorithms.

3.6 Formal Properties of WIDL-NGLM Algorithms

The following theorem states the correctness of the proposed WIDL-NGLM algorithms. Algorithm WIDL-NGLM-A* comes with the strong guarantee that it finds the maximum probability path encoded by a WIDL-graph under a log-linear combination of the specified WIDL

probability distribution and n-gram language model probability distributions. At the same time, algorithm WIDL-NGLM-BEAM comes with a weaker guarantee of finding the maximum probability path, but allows for a faster search while maintaining good approximations for the best path.

Theorem 4 *Let ω be a WIDL-expression, γ_ω its WIDL-graph, and A_ω^{n-1} its corresponding pFSA under n-gram language models. Let $\overline{h} = \langle \gamma_\omega, LM_1 \ldots LM_M \rangle$ (LM_i an n-gram language model, $1 \leq i \leq M$), $\overline{\lambda} = \langle \lambda_0, \lambda_1, \ldots, \lambda_M \rangle$ be a vector of real numbers, and b also a real number. Algorithm* WIDL-NGLM-A$^*(\gamma_\omega, \overline{LM}, \overline{\lambda})$ *finds the path of maximum probability under Equation 3.2. Algorithm* IDL-NGLM-BEAM$(\gamma_\omega, \overline{LM}, \overline{\lambda}, b)$ *finds the path of maximum probability under Equation 3.2, if all states in A_ω^{n-1} along this path are selected by its* BEAMSTATES *function when using b as its beam parameter.*

Proof: One should first note that finding the path of maximum probability under Equation 3.2 is equivalent (via its second formulation) with finding the path of minimum cost, where cost is computed as the negative logarithm of the probability value. By the definition of the FE_S and FCE_S terms, Equation 3.4 guarantees that, for any state S in A_ω^{n-1}, cost $h_\omega(S)$ is *admissible*, that is, it is an underestimation of the actual cost to arrive at the final state from S. The unfolding mechanism from algorithm WIDL-NGLM-A*, based on the priority queue Q, guarantees that the states of A_ω^{n-1} are built in *monotonically increasing* order of costs.

Together, these two conditions fulfill the required properties needed to apply the A* correctness result from (Russell & Norvig 1995), from which the correctness of the path found by the WIDL-NGLM-A* algorithm directly follows.

To prove the correctness of the WIDL-NGLM-BEAM algorithm, I use the hypothesis condition, from which it follows that, at each iteration i, the prefix of length i of the minimum cost path is in the $active_i$ set of states. Consider now that the actual length of the minimum cost path is N. As the algorithm finishes when the lowest scoring $unfold$ed state is also final, it follows that the algorithm finishes after $T \geq N$ steps, and queue B contains the final state that points to the minimum cost path (inserted in B at iteration N). After the sorting step, the state that points to the minimum cost path becomes the first state in B, and it is returned as the output of the algorithm. □

The next lemma relates the size of set UNFOLD_ω^n to the size of set $n\text{-}\mathsf{cut}(\omega)$. As n is held constant throughout the discussion regarding the run-time complexity, I will abuse notation and simply write UNFOLD_ω and $\mathsf{cut}(\omega)$. This result is used by the theorem presented next, which characterizes the run-time of the proposed algorithms.

Lemma 5 *Let Σ and Δ be finite alphabets, ω be an WIDL-expression over Σ and Δ, and $\gamma_\omega = (V, E, v_s, v_e, \lambda, r)$ be the associated WIDL-graph. Let also $k = \mathsf{width}(\omega)$. Then*

$$\mathrm{O}(|\text{UNFOLD}_\omega|) = \mathrm{O}(k|\mathsf{cut}_\omega|).$$

The proof for this lemma is given as a formal fragment, as it is not necessary for the understanding of the results concerning the run-time of the WIDL-NGLM algorithms.

For the proof of Lemma 5, I use induction on $N = \mathsf{nop}(\omega)$ (see Section 2.4, Definition 10). For $N = 0$, $\omega = a$, $a \in \Sigma$. In this case $k = 1$, $|\text{UNFOLD}_\omega| = 1$, and $|\mathsf{cut}_\omega| = 2$, and the lemma is true.

Consider now an WIDL-expression ω with $\mathsf{nop}(\omega) = N$. The induction hypothesis (which I will further refer to as IH1) states that the result of the lemma is true for all ω' with $\mathsf{nop}(\omega') < N$. I distinguish the following cases:

- $\omega = \vee_\delta(\omega_1, \ldots, \omega_n)$, $\mathrm{O}(|\text{UNFOLD}_{\omega_i}|) = \mathrm{O}(k_i|\mathsf{cut}_{\omega_i}|), k_i = \mathsf{width}(\omega_i), 1 \leq i \leq n$. In this case $k = \mathsf{max}_i(k_i)$, and $|\text{UNFOLD}_\omega| = \Sigma_i 2 + |\text{UNFOLD}_{\omega_i}|$, $|\mathsf{cut}_\omega| = 2 + \Sigma_i|\mathsf{cut}_\omega|$. Then,
  - $\mathrm{O}(|\text{UNFOLD}_\omega|) = \mathrm{O}(\Sigma_i\ 2 + |\text{UNFOLD}_{\omega_i}|) = \mathrm{O}(\Sigma_i|\text{UNFOLD}_{\omega_i}|) = \Sigma_i\mathrm{O}(|\text{UNFOLD}_{\omega_i}|) = \Sigma_i\mathrm{O}(k_i|\mathsf{cut}_{\omega_i}|) = \mathrm{O}(\Sigma_i k_i|\mathsf{cut}_{\omega_i}|) = \mathrm{O}(k|\mathsf{cut}_\omega|)$.
- $\omega = \times(\omega')$, and $\mathrm{O}(|\text{UNFOLD}_{\omega'}|) = \mathrm{O}(k'|\mathsf{cut}_{\omega'}|)$, with $k' = \mathsf{width}(\omega')$. Then $k = k'$, $|\text{UNFOLD}_\omega| = |\text{UNFOLD}'_\omega| + 2$, $|\mathsf{cut}_\omega| = |\mathsf{cut}'_\omega| + 2$, and therefore
  - $\mathrm{O}(|\text{UNFOLD}_\omega|) = \mathrm{O}(|\text{UNFOLD}_{\omega'}|) = \mathrm{O}(k'|\mathsf{cut}_{\omega'}|) = \mathrm{O}(k|\mathsf{cut}_\omega|)$.
- $\omega = \omega_1 \cdot \omega_2$, and $\mathrm{O}(|\text{UNFOLD}_{\omega_i}|) = \mathrm{O}(k_i|\mathsf{cut}_{\omega_i}|), 1 \leq i \leq 2$. In this case $k = \mathsf{max}(k_1, k_2)$, and $|\text{UNFOLD}_\omega| = |\text{UNFOLD}_{\omega_1}| + |\text{UNFOLD}_{\omega_2}| + 1$, $|\mathsf{cut}_\omega| = |\mathsf{cut}_{\omega_1}| + |\mathsf{cut}_{\omega_2}|$, and therefore
  - $\mathrm{O}(|\text{UNFOLD}_\omega|) = \mathrm{O}(|\text{UNFOLD}_{\omega_1}| + |\text{UNFOLD}_{\omega_2}) = \mathrm{O}(k_1|\mathsf{cut}_{\omega_1}| + k_2|\mathsf{cut}_{\omega_2}|) = \mathrm{O}(k|\mathsf{cut}_\omega|)$.
- $\omega = \|_\delta(\omega_1, \ldots, \omega_n)$, $\mathrm{O}(|\text{UNFOLD}_{\omega_i}|) = \mathrm{O}(k_i|\mathsf{cut}_{\omega_i}|), k_i = \mathsf{width}(\omega_i), 1 \leq i \leq n$. This is the most interesting case. Without losing the generality of the proof, I will assume in what follows that the number of arguments for $\|$ and $\vee$ is $i = 2$. In

order to prove the lemma result for $\omega = \|_\delta(\omega_1, \omega_2)$, I use induction on the number $N_2 = \mathsf{nop}(\omega_2)$.

I prove first the lemma for the base case $\omega = \|_\delta(\omega_1, a)$, that is, $\mathsf{nop}(\omega_2) = 0$. For this case, $k_1 = \mathsf{width}(\omega_1)$ and $k = k_1 + 1$, or $k_1 = k - 1$. The set cut_ω is the set of vertices of the finite automaton corresponding to expression ω of width k (examples of cut sets are the sets of vertices in Figure 3.5, (a)-(c)); I denote its cardinality by V_k. The set UNFOLD_ω is the set of edges of the finite automaton corresponding to expression ω of width k (examples of UNFOLD sets are the sets of edges in Figure 3.5, (a)-(c)); I denote its cardinality by E_k.

From the V_{k-1} vertices corresponding to ω_1, the set cut_ω with cardinality V_k is obtained by adding, to each cut element, either the start vertex or the end vertex of the a transition. For example, in Figure 3.5-(a), s_a and e_a are added to the cuts c_1 and c_2 of ω_1. I therefore have twice the number of cuts from cut_{ω_1} in cut_ω, or

$$V_k = 2V_{k-1}. \tag{3.5}$$

Analogously, from the E_{k-1} edges corresponding to ω_1, the set UNFOLD_ω with cardinality E_k is obtained by including twice E_{k-1} edges (the ones between the cuts before the transition a is made, and the ones between the cuts after the transition a is made, see Figure 3.5, (a)-(c)), plus one edge (corresponding to transition a) for each pair of cuts in cut_ω that is identical modulo the start and end vertex of transition a. There are exactly V_{k-1} such pairs of cuts, so I can write

$$E_k = 2E_{k-1} + V_{k-1}. \tag{3.6}$$

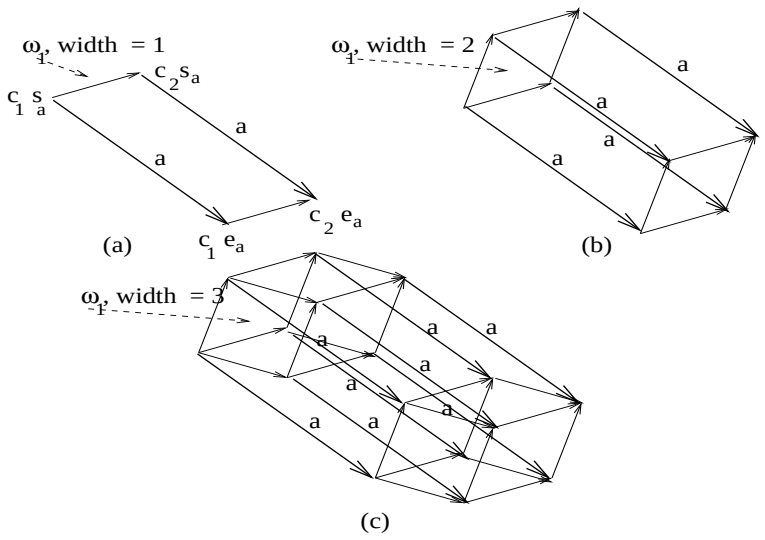

Figure 3.5: The probabilistic finite-state automata corresponding to expressions of the type $\|_\delta(\omega_1, a)$, with $\mathsf{width}(\omega_1) = 1, 2$ and 3, in (a), (b), and (c), respectively.

Then $V_1 = 2$ and $E_1 = 1$, and therefore the system of equations 3.5 and 3.6 can be written in closed form as

$$V_k = 2^k \quad E_k = k2^{k-1} \tag{3.7}$$

From equation 3.7, $E_k = \frac{k}{2}V_k$, which means $|\textsc{unfold}_\omega| = \frac{k}{2}|\mathsf{cut}_\omega|$, and therefore $\mathrm{O}(|\textsc{unfold}_\omega|) = \mathrm{O}(k|\mathsf{cut}_\omega|)$.

Suppose now that $\mathsf{nop}(\omega_2) = N_2$. The induction hypothesis (which I will further refer to as IH2) states that the result of the lemma is true for $\omega = \|_\delta(\omega_1, \omega_2')$ for all ω_2' with $\mathsf{nop}(\omega_2') < N_2$. The following cases are possible:

- $\omega_2 = \vee_{\delta_2}(\omega_2', \omega_2'')$. Then $\omega = \|_\delta(\omega_1, \vee_{\delta_2}(\omega_2', \omega_2'')) = \vee_{\delta_2}(\|_{\delta'}(\omega_1, \omega_2'), \|_{\delta''}(\omega_1, \omega_2''))$. Because $\mathsf{nop}(\omega_2') < N_2$ and $\mathsf{nop}(\omega_2'') < N_2$, IH2 implies that the lemma holds for the argument expressions $\|_{\delta'}(\omega_1, \omega_2')$ and $\|_{\delta''}(\omega_1, \omega_2'')$. Because $\mathsf{nop}(\|_{\delta'}(\omega_1, \omega_2')) < N$ and $\mathsf{nop}_{\delta''}(\|(\omega_1, \omega_2'')) < N$, IH1 implies that the lemma holds for $\omega = \|_\delta(\omega_1, \omega_2)$.

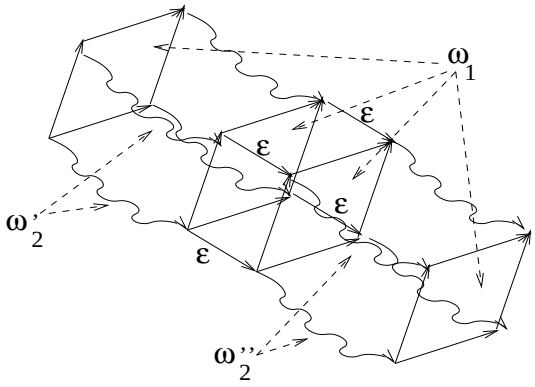

Figure 3.6: The probabilistic finite-state automaton corresponding to $\|_\delta(\omega_1, \omega_2' \cdot \omega_2'')$, with $\mathsf{width}(\omega_1) = 2$.

- $\omega_2 = \|_{\delta_2}(\omega_2', \omega_2'')$. Then $\omega = \|_\delta(\omega_1, \|_{\delta_2}(\omega_2', \omega_2'')) = \|_{\delta'}(\omega_1, \omega_2', \omega_2'')$. Because $\mathsf{nop}(\omega_2') < N_2$ and $\mathsf{nop}(\omega_2'') < N_2$, IH2 implies that the lemma holds for $\omega = \|_\delta(\omega_1, \omega_2)$.

- $\omega_2 = \times(\omega_2')$. At this point, one should note that choosing to perform the second induction on argument ω_2 was arbitrary, and the same proofs hold for an induction on argument ω_1. This means that the same proofs cover all the cases for the structure of ω_1, except when $\omega_1 = \times(\omega_1')$. In this case, $\omega = \|_\delta(\times(\omega_1'), \times(\omega_2')) = \vee_{\delta'}(\omega_1' \cdot \omega_2', \omega_2' \cdot \omega_1')$. Because $\mathsf{nop}(\vee_{\delta'}(\omega_1' \cdot \omega_2', \omega_2' \cdot \omega_1')) < N$, IH1 implies that the lemma holds for $\omega = \|_\delta(\omega_1, \omega_2)$.

- $\omega_2 = \omega_2' \cdot \omega_2''$. This case is illustrated in Figure 3.6 for the case when $\mathsf{width}(\omega_1) = 2$. Let c' and c'' be the cardinality of the sets $\mathsf{cut}_{\omega_2'}$ and $\mathsf{cut}_{\omega_2''}$, respectively, and let $c = c' + c''$. Let also d' and d'' be the cardinality of the sets $\text{UNFOLD}_{\omega_2'}$ and $\text{UNFOLD}_{\omega_2''}$, respectively.

  I treat this case similarly to the base case for the second induction. Let $k = \mathsf{width}(\omega)$, let V_k be the cardinality of the set cut_ω, and let E_k be the cardinality

of the set UNFOLD_ω. Let $k_1 = \mathsf{width}(\omega_1)$, and let $k_2 = \mathsf{width}(\omega_2)$. Then $k = k_1 + k_2$, with $k > k_1$ (because $k_2 \geq 1$).

From the c' and c'' vertices corresponding to ω_2' and ω_2'', the set cut_ω with cardinality V_k is obtained by adding, to each cut element in cut_{ω_1}, one vertex from either $\mathsf{cut}_{\omega_2'}$ or $\mathsf{cut}_{\omega_2''}$. I therefore have

$$V_k = cV_{k_1}. \tag{3.8}$$

From the d' and d'' edges corresponding to ω_2' and ω_2'', the set UNFOLD_ω with cardinality E_k is obtained by including c times each relation from UNFOLD_{ω_1}, plus one edge for each ϵ transition between the cuts of $\mathsf{cut}_{\omega_2'}$ and $\mathsf{cut}_{\omega_2''}$ (see Figure 3.6). There are exactly V_{k_1} such ϵ transitions, so I can write

$$E_k = cE_{k_1} + V_{k_1}. \tag{3.9}$$

The system of equations 3.8 and 3.9 can be written in closed form as

$$V_k = c^k \quad E_k = (k + c^{k_2})c^{k-k_2} \tag{3.10}$$

This means that $E_k = (1+\frac{k}{c^{k_2}})V_k$, and therefore $\text{O}(|\text{UNFOLD}_\omega|) = \text{O}(k|\mathsf{cut}_\omega|)$.

□

The next theorem characterizes the run-time complexity of algorithms WIDL-NGLM-A* and WIDL-NGLM-BEAM, in terms of an input WIDL-expression ω and its corresponding

WIDL-graph γ_ω complexity. There are two factors that linearly influence the run-time complexity of these algorithms: k is the maximum number of nodes in γ_ω needed to represent a state in A_ω, defined as $\mathsf{width}(\omega)$ (Section 2.4, Definition 8) – k depends solely on ω; and K is the number of states explicitly built for A_ω^{n-1}, the pFSA corresponding to γ_ω under n-gram language models – K depends on ω and n, the length of the context used by the n-gram language models.

Theorem 6 *Let ω be a WIDL-expression and $\gamma_\omega = (V, E, v_s, v_e, \lambda, r)$ its WIDL-graph. Let $k = \mathsf{width}(\omega)$, and $K = \Sigma_i |active_i|$. Algorithm* WIDL-NGLM-A* *has run-time complexity $\mathrm{O}(kK + K \log K)$, and algorithm* WIDL-NGLM-BEAM *has run-time complexity $\mathrm{O}(kK)$. Moreover, K is characterized in terms of γ_ω by the following tight upperbound: $K \leq \left(\frac{|V|}{k}\right)^k$.*

Proof: For the WIDL-NGLM-A*, one should first note that the size of the $unfold$ and $active$ sets is always one. If $k = \mathsf{width}(\omega)$, adding new elements to the relation $\textsc{unfold}_\omega$ takes time O(k) (Lemma 5). Therefore, steps 4-6 take time O(k) per iteration. As there are $K = \Sigma_i |active_i|$ iterations of the **while** loop, it means $\mathrm{O}(kK)$ time for all steps 4-6. The queue management (step 7) implies, for a priority queue implemented as a heap, an upperbound of $\mathrm{O}(\log K)$ (for the heapify operation) per iteration, and therefore $\mathrm{O}(K \log K)$ for all steps 7. The total runtime of the WIDL-NGLM-A* algorithm is therefore $\mathrm{O}(kK + K \log K)$.

For the WIDL-NGLM-BEAM, if I denote by $K_i = |active_i|$ at iteration i in the **while** loop, I have $\mathrm{O}(kK_{i+1})$ for steps 4-6 (as $active_{i+1} = unfold_i$). As $K = \Sigma_i K_i$, the total runtime for the entire **while** loop is $\mathrm{O}(kK)$. As the number of final states (size of queue B) depends only on n, the length of the context used by the n-gram language models, the sort

operation takes constant time $\mathrm{O}(1)$ with respect to the input expression. The overall runtime of the WIDL-NGLM-BEAM is therefore $\mathrm{O}(kK)$.

From Lemma 1, it follows that there exists an IDL-expression π such that $|\mathsf{cut}(\omega)| = |\mathsf{cut}(\pi)|$. From Nederhof and Satta (2004a), it follows that $|\mathsf{cut}(\pi)| \leq \left(\frac{|V|}{k}\right)^k$. As, in the worst case, $K = |\mathsf{cut}(\omega)|$, it follows that $K \leq \left(\frac{|V|}{k}\right)^k$. $\square$

For a WIDL-expression ω, for which γ_ω has $|V|$ nodes and $\mathsf{width}(\omega) = k$, the quantity $\left(\frac{|V|}{k}\right)^k$ is said to characterize the complexity of ω. The fact that the run-time behavior of these algorithms is linear in the complexity of the input WIDL-expression (with an additional log factor for priority management in the A* version) allows us to say that these algorithms are efficient with respect to the task they accomplish. However, depending on the input WIDL-expression ω, the run-time of our algorithms intersecting ω with n-gram language models can vary in complexity from linear to exponential. That is, for a WIDL-expression $\omega = \|_u(w_1, \ldots, w_n)$, $u = \{\mathsf{perms} \overset{\mathsf{uniform}}{\rightarrow} 1\}$ (uniform distribution over a bag of n words), one has $K \leq (\frac{2n}{n})^n = 2^n$ (the factor of 2 comes getting two nodes in γ_ω for each argument of $\|_u$), and therefore a $\mathrm{O}(n2^n)$ complexity. This exponential complexity comes as no surprise given that the problem of intersecting an n-gram language model with a bag of words is known to be NP-complete (Knight 1999). On the other hand, for a WIDL-expression $\omega = w_1 \cdot \ldots \cdot w_n$ (sequence of n words), one has $K = n$, and therefore an $\mathrm{O}(n)$ generation algorithm.

In general, for WIDL-expressions for which k is bounded, algorithms WIDL-NGLM-A* and WIDL-NGLM-BEAM perform in $\mathrm{O}(n^k \log n)$ and $\mathrm{O}(n^k)$, respectively. That is, they have a polynomial time in the number of words available for generation.

3.7 An Intrinsic Evaluation of WIDL-NGLM Algorithms

In this section, I present results concerning the performance of the WIDL-NGLM algorithms on a word-ordering task. This task can be easily defined as follows: from a bag of words originating from some sentence, reconstruct the original sentence as faithfully as possible. As no information regarding the original word order in the sentence is preserved, the input can be represented as WIDL-expressions using a $\|_\delta$ operator over all the words in the bag, with δ uniform distribution.

As an example, from a sentence such as the gifts are donated by american companies, I create the WIDL-expression

$$\langle s \rangle \cdot \|_u(\mathsf{the}, \mathsf{gifts}, \mathsf{donated}, \mathsf{companies}, \mathsf{by}, \mathsf{are}, \mathsf{american}) \cdot \langle /s \rangle, \quad u = \{\mathsf{perms} \overset{\mathsf{uniform}}{\rightarrow} 1\},$$

for which a possible realization is donated by the american companies are gifts. Using the precedence $(\cdot)$ operator, it is straightforward to encode beginning and end of sentence boundaries in the WIDL-expressions, using the $\langle s \rangle$ and $\langle /s \rangle$ symbols, respectively. Since this is generation from bag-of-words, the task is known to be at the high-complexity extreme of the run-time behavior of the WIDL-NGLM algorithms. Therefore, I consider it a good test for the ability of these algorithms to scale up to increasingly complex inputs.

I use the state-of-the-art, publicly available SRI Language Model Toolkit to train a trigram language model using Kneser-Ney smoothing, on 10 million sentences (170 million words) from the Wall Street Journal (WSJ), lower case and no final punctuation. The test data is also

lower case (such that upper-case words cannot be hypothesized as first words), with final punctuation removed (such that periods cannot be hypothesized as final words), and consists of 2000 unseen WSJ sentences of length 3-7, and 2000 unseen WSJ sentences of length 10-25.

The algorithms I tested in this experiments are the ones presented in Section 3.4, plus two baseline algorithms. The first baseline algorithm, L, uses an inverse-lexicographic order for the bag items as its output, in order to get the word *the* on sentence initial position. The second baseline algorithm, G, is a greedy algorithm that realizes sentences by maximizing the probability of joining any two word sequences until only one sequence is left.

For the A* algorithm, an admissible cost is computed for each state S in the corresponding pFSA, as the sum (over all unused words) of the minimum language model cost combination (i.e., maximum probability) of each unused word when conditioning over all sequences of two words available at that particular state for future conditioning (see Equation 3.4, with $FE_S = FE_S^{all}$). These estimates are also used by the beam algorithm for deciding which WIDL-graph nodes are not unfolded. For the A* algorithms, I use the notation A$^*_k$ to denote the k-steps greedy version. For the beam algorithms, I use the notation B$_p$ to specify a probabilistic beam of size p.

My first batch of experiments concerns bags-of-words of size 3-7, for which exhaustive search is possible. In Table 3.1, I present the results on the word-ordering task achieved by various algorithms. I evaluate accuracy performance using two automatic metrics: an identity metric, ID, which measures the percent of sentences recreated exactly, and BLEU (Papineni *et al.* 2002), which gives the geometric average of the number of uni-, bi-, tri-, and four-grams

ALG	ID (%)	BLEU	Search Errors (%)	Unfold (%)	Speed (sec./bag)
L	2.5	9.5	97.2 (95.8)	N/A	.000
G	30.9	51.0	67.5 (57.6)	N/A	.000
BFS	67.1	79.2	0.0 (0.0)	100.0	.072
A^*	67.1	79.2	0.0 (0.0)	12.0	.010
A^*_1	60.5	74.8	21.1 (11.9)	3.2	.004
A^*_2	64.3	77.2	8.5 (4.0)	5.3	.005
$B_{0.2}$	65.0	78.0	9.2 (5.0)	7.2	.006
$B_{0.1}$	66.6	78.8	3.2 (1.7)	13.2	.011

Table 3.1: Bags-of-words of size 3-7: accuracy (ID, BLEU), Search Errors (and Estimated Search Errors), space savings (Unfold), and speed results.

recreated exactly. I evaluate the search performance by the percent of Search Errors made by these algorithms, as well as a percent figure of Estimated Search Errors, computed as the percent of searches that result in a string with a lower probability than the probability of the original sentence. To measure the impact of using WIDL-expressions for this task, I also measure the percent of unfolding of a WIDL graph with respect to a full unfolding. I report speed results as the average number of seconds per bag-of-words, when using a 3.0GHz CPU machine under a Linux OS.

The first notable result in Table 3.1 is the savings achieved by the A^* algorithm under the WIDL representation. At no cost in accuracy, it unfolds only 12% of the edges, and achieves a 7 times speed-up, compared to the BFS algorithm. The savings achieved by not unfolding are especially important, since the exponential complexity of the problem is hidden by the WIDL representation via the folding mechanism of the $\|$ operator. The algorithms that find

ALG	ID (%)	BLEU	Est. Search Errors (%)	Speed (sec./bag)
L	0.0	1.4	99.9	0.0
G	1.2	31.6	83.6	0.0
A_1^*	5.8	47.7	34.0	0.7
A_2^*	7.4	51.2	21.4	9.5
$B_{0.2}$	9.0	52.1	23.3	7.1
$B_{0.1}$	12.2	52.6	19.9	36.7

Table 3.2: Bags-of-words of size 10-25: accuracy (ID, BLEU), Estimated Search Errors, and speed results.

sub-optimal solutions also perform well. While maintaining high accuracy, the A_2^* and $B_{0.2}$ algorithms unfold only about 5-7% of the edges, at 12-14 times speed-up.

My second batch of experiments concerns bag-of-words of size 10-25, for which exhaustive search is no longer possible (Table 3.2). Not only exhaustive search, but also full A^* search is too expensive in terms of memory (2GiB of RAM) and speed. Only the greedy versions A_1^* and A_2^*, and the beam search using tight probability beams (0.2-0.1) scale up to these bag sizes. Because I no longer have access to the string of maximum probability, I report only the percent of Estimated Search Errors. In terms of accuracy, there are around 20% Estimated Search Errors for the best performing algorithms (A_2^* and $B_{0.1}$), which means that 80% of the time the algorithms are able to find sentences of equal or better probability than the original sentences.

3.8 Conclusions

In this chapter, I have shown that the WIDL formalism allows for the formulation of efficient algorithms for intersecting, scoring, and ranking candidate realizations using probabilistic language models based on n-grams. I have presented theoretical results concerning the correctness and efficiency of these algorithms.

Because the requirements for producing WIDL-expressions are minimal, a WIDL-based sentence realizer implementing the algorithms presented in this chapter can be directly employed in text-to-text NLG applications where only application-specific mechanisms have been employed to date. As a proof-of-concept, I have presented in this chapter empirical results that show that these algorithms scale up to handling WIDL-expressions of high complexity.

Chapter 4

Automatic Headline Generation using WIDL-expressions

4.1 Previous Approaches to Headline Generation

Two main approaches have been proposed to date for solving the headline generation task. They can be characterized as either extractive, top-down approaches, or abstractive, bottom-up approaches to headline generation.

4.1.1 Extractive Approaches

Headline generation systems that use the extractive approach generally implement the following steps. First, the most informative sentence in the given document is identified. For the news genre, in particular, the first sentence of the document is usually the most informative. Second, some form of sentence compression is performed, such that the headline meets some length requirement, usually set to 10 words. Known compression algorithms use either symbolic, rules-based approaches (Chandrasekar, Doran, & Bangalore 1996; Jing & McKeown 1999; Canning

et al. 2000; Dorr, Zajic, & Schwartz 2003), or stochastic, supervised approaches (Knight & Marcu 2002).

Symbolic approaches to compression, operating on syntactic sentence structures (Dorr, Zajic, & Schwartz 2003), usually produce correct and grammatical sentences, but tend to shrink the sentences too much, and in the process lose important content information. To remedy this problem, another step is added, in which the compressed sentence is "padded" with important content words or phrases. Finding this content information can be achieved by computing noun phrase coreference chains (Bergler *et al.* 2003) across the input document, or by discovering "topics" using Unsupervised Topic Discovery (Schwartz 2001).

In one specific proposal (Zajic, Dorr, & Schwartz 2004), an algorithm that identifies "topic keywords" is run over the input document. The retrieved keywords are generally indicative of the topic of the document, and carry important content information. Therefore, the information content of the output headline is increased by adding to the compressed sentence the topic keywords that are not already present. One should note that this system, called Topiary, has provided the best performance at the last headline generation competition, which was run by the National Institute of Standards and Technology (NIST) in 2004.

I call these approaches extractive, because they build their output around a sentence extracted from the input document. Their advantage is that the fluency of the output is ensured by the fact that the extracted sentence is produced by a human (the author of the input document). The disadvantage is that the coverage of their output may be limited. Also, the extractive

approach might not prove suitable for building headlines outside the news genre, where the journalistic convention of rendering the most important information first does not apply.

4.1.2 Abstractive Approaches

In contrast, abstractive approaches create headlines in a bottom-up manner, starting from important, individual words and phrases, that are glued together to create a fluent sentence. Inspired from work in Machine Translation (Brown *et al.* 1993), a system developed by Banko et al. (2000) generates headlines using statistical models for content selection and sentence realization. Another abstractive headline generation system, created by Zhou and Hovy (2003), runs first an algorithm that identifies a list of "keywords". The retrieved keywords are then used to extract phrases from the document, which are linked together to create headlines. The advantage of this approach is that these keywords carry a lot of the content information of the document, and therefore a headline that contains many such keywords is likely to be informative. The disadvantage is that it is difficult to create fluent outputs in this manner.

A more recent abstractive headline generation system, proposed by Wan et al. (2005), starts from dependency structures extracted from an input document, and glues them together using n-grams to smooth the transitions between adjacent dependency structures. More general fusion techniques, such as Information Fusion (Barzilay 2003), are also suitable for abstractive headline generation.

The approach to headline generation described in this chapter combines some of the ideas of the previous works described. It uses a list of "keywords" to identify the words that carry

most of the content in an input document. It also uses syntactic information, extracted from parse trees projected over the sentences of the input document, to build syntactically-driven phrases around the extracted keywords. Using these sources of information, it employs WIDL-expressions to encode probabilistically content biases present in the input document, and the WIDL framework allows us to employ a generic natural language generation system in order to create headlines that are both fluent and informative.

4.2 Automatic Creation of Headline-specific WIDL-expressions

The headline generation system I describe in this chapter is built using the architecture presented in Figure 1.1 (Chapter 1). The generation engine implements the generation algorithms described in Chapter 3. In this section, I present the algorithm used to implement a headline-specific WIDL-expression creation module, which automatically creates WIDL-expressions starting from input documents.

First, I use the algorithm of Zhou and Hovy (2003) to extract a weighted list of topic keywords from an input document (see the example in Figure 4.1(a), for which the input is a document describing incidents at the Turkey-Iraq border and the surrounding region). Next, I parse the input document using my own implementation of Collins's parser (1999), extract the lexical dependencies of all topic keywords, and create a phrase list for each topic keyword. For each such phrase list, I associate a probability distribution computed using the frequency of these phrases (I assume that higher frequency is indicative of increased importance) and the position of these phrases within the document (if the phrase occurs multiple times, the first occurrence

(a) Topic Keywords

iraq 0.32, syria 0.25, rebels 0.22, kurdish 0.17, turkish 0.14, attack 0.10

(b) Lexical-dependency Phrases

iraq	northern iraq 0.48, in iraq 0.39, iraq and iran 0.12
syria	into syria 0.55, and syria 0.45
rebels	attacked rebels 0.63, rebels fighting 0.28, kurdish rebels 0.09
turkish	turkish government 0.77, turkish soldiers 0.23

(c) WIDL-expression

$$
\begin{aligned}
\|_{\delta_0}(\quad & \vee_{\delta_1}(\times(\text{northern} \cdot \text{iraq}),\ \times(\text{in} \cdot \text{iraq}),\ \times(\text{iraq} \cdot \text{and} \cdot \text{iran}),\ \text{eps}\ ), \\
& \vee_{\delta_2}(\times(\text{into} \cdot \text{syria}),\ \times(\text{and} \cdot \text{syria}),\ \text{eps}\ ), \\
& \vee_{\delta_3}(\times(\text{attacked} \cdot \text{rebels}),\ \times(\text{rebels} \cdot \text{fighting}),\ \times(\text{kurdish} \cdot \text{rebels}),\ \text{eps}\ ), \\
& \vee_{\delta_4}(\times(\text{turkish} \cdot \text{government}),\ \times(\text{turkish} \cdot \text{soldiers}),\ \text{eps}\ ) \\
)\ & \\
& \delta_0 = \{\text{perms} \overset{\text{uniform}}{\rightarrow} 1\} \\
& \delta_1 = \{1 \rightarrow 0.48,\ 2 \rightarrow 0.39,\ 3 \rightarrow 0.12,\ 4 \rightarrow 0.01\} \\
& \delta_2 = \{1 \rightarrow 0.51,\ 2 \rightarrow 0.40,\ 3 \rightarrow 0.09\} \\
& \delta_3 = \{1 \rightarrow 0.63,\ 2 \rightarrow 0.28,\ 3 \rightarrow 0.09,\ 4 \rightarrow 0.008\} \\
& \delta_4 = \{1 \rightarrow 0.75,\ 2 \rightarrow 0.21,\ 3 \rightarrow 0.05\}
\end{aligned}
$$

(d) Output headline (most likely realization)

TURKISH GOVERNMENT ATTACKED REBELS IN IRAQ AND SYRIA

Figure 4.1: From an input document, a weighted list of topic-keywords is extracted(a), from which phrases are proposed (b) and combined in a WIDL-expression (c). The output (d) for our automatic headline generation system is also shown.

is considered to determine position; I assume that proximity to the beginning of the document is also indicative of importance). For the current running example, this list is presented in Figure 4.1(b).

The algorithm which produces WIDL-expressions combines first the lexical-dependency phrases for each topic keyword i under a $\vee_{\delta_i}$ operator. Additionally, a special eps expression is also specified as an argument for each $\vee_{\delta_i}$ operator, which accounts for the possibility of

dropping topic i. (The eps expression is treated as a regular Σ-label within the WIDL formalism, but it is filtered out from the output headline.) For a given topic keyword, the probability values associated with each phrase are used to specify the probability associated with each argument of the δ_i specification, after setting aside a small probability mass for the eps choice. The probability of the eps choice is computed such that its cost (i.e., the $-\log$ of the probability) is twice the cost of the most expensive non-eps choice. All of the $\vee_{\delta_i}$-headed expressions are finally combined into a single WIDL-expression using a $\|_{\delta_0}$ operator, $\delta_0 = \{\mathsf{perms} \stackrel{\mathsf{uniform}}{\rightarrow} 1\}$ (uniform probability over all the possible orders of the keyword phrases). The WIDL-expression in Figure 4.1(c) results from applying this algorithm to the phrases in our example. On average, a WIDL-expression created by this algorithm, using $m = 6$ keywords and an average of $k = 4$ lexical-dependency phrases per keyword, compactly encodes a candidate set of about 3 million possible realizations. As the specification of the $\|_{\delta_0}$ operator takes space O(1), Theorem 2 (Chapter 2) guarantees that the space complexity of these expressions is O(mk).

Finally, I generate headlines from WIDL-expressions using the WIDL-NGLM-BEAM^{100} algorithm (beam-search algorithm with histogram beam set to 100), which interpolates the probability distributions represented by the WIDL-expressions with n-gram language model distributions. The output presented in Figure 4.1(d) is the most likely headline realization produced by this algorithm.

4.3 Headline Generation Evaluation

To evaluate the accuracy of the WIDL-based headline generation system, I use the documents from the DUC 2003 evaluation competition. Half of these documents are used as development set (283 documents), and the other half is used as test set (273 documents). I automatically measure performance by comparing the produced headlines against one reference headline produced by a human using ROUGE_2 (Lin 2004).

For each input document, I train and use two trigram language models, using the SRI Language Model Toolkit (with modified Kneser-Ney smoothing). A general trigram language model, trained on 170M English words from the Wall Street Journal, is used to model fluency. A document-specific trigram language model, trained on-the-fly for each input document, accounts for both fluency and content validity. I also use a word-count model (which counts the number of words in a proposed realization) and a phrase-count model (which counts the number of phrases in a proposed realization), which allow one to learn to produce headlines that have restrictions in the number of words allowed (10, in this case). Note that count models of this type can be easily integrated with n-gram models under WIDL-expressions, by adding to the n-history string of state S (Chapter 2, Section 2.3) an additional token recording the length of the longest path from the initial state to state S. In summary, I use Equation 3.2 (Chapter 3) with feature function h_0 for the WIDL-expressions, two feature functions h_1 and h_2 for the two trigram language models, and two feature functions h_3 and h_4 for the word-count and phrase-count model, respectively. The interpolation weights $\overline{\lambda}$ (Equation 3.2) are trained using

ALG	♯(UNI)	♯(BI)	AV. LEN.	ROUGE_1	**ROUGE_2**
Extractive					
Lead10	458	114	9.9	20.8	11.1
HedgeTrimmer†	399	104	7.4	18.1	9.9
Topiary‡	576	115	9.9	26.2	12.5
Abstractive					
Keywords	585	22	9.9	26.6	5.5
Webcl	311	76	7.3	14.1	7.5
WIDL-NGLM-BEAM^{100}	562	126	10.0	25.5	**12.9**

Table 4.1: Headline generation evaluation. I compare extractive algorithms against abstractive algorithms, including a WIDL-based headline generation algorithm.

discriminative training (Och 2003) using ROUGE_2 as the objective function, on the development set.

The results are presented in Table 4.1. I compare the performance of several extractive algorithms against several abstractive algorithms (see Section 4.1). For the extractive algorithms, **Lead10** is a baseline which simply proposes as headline the first 10 words of the lead sentence. **HedgeTrimmer**† is our implementation of the Hedge Trimer algorithm (Dorr, Zajic, & Schwartz 2003), and **Topiary**‡ is our implementation of the Topiary algorithm (Zajic, Dorr, & Schwartz 2004). For the abstractive algorithms, **Keywords** is a baseline that proposes as headline the sequence of topic keywords sorted according to their scores (from highest to lowest), **Webcl** is an algorithm that creates headlines as described in (Zhou & Hovy 2003), and WIDL-NGLM-BEAM^{100} is the histogram beam–search algorithm described in Chapter 3, Section 3.4.

The results of this experiment show that the WIDL-based approach to generation is capable of creating headlines that comparable favorably, in both content and fluency, with extractive,

THREE GORGES PROJECT IN CHINA HAS WON APPROVAL
WATER IS LINK BETWEEN CLUSTER OF E. COLI CASES
SRI LANKA 'S JOINT VENTURE TO EXPAND EXPORTS
OPPOSITION TO EUROPEAN UNION SINGLE CURRENCY EURO
OF INDIA AND BANGLADESH WATER BARRAGE

Figure 4.2: Headlines generated automatically using a WIDL-based sentence realization system.

state-of-the-art results (Zajic, Dorr, & Schwartz 2004). The best performance is obtained by the WIDL-NGLM-BEAM^{100} system, 12.9 ROUGE$_2$ score, followed by the system implementing the Topiary solution, 12.5 ROUGE$_2$ score. The difference between these two scores is not statistically significant at 95% confidence, but it is statistically significant compared to all the other scores presented in Table 4.1. One should also note that this evaluation was conducted such that the comparisons are more direct and meaningful, as the compared implementations use the same algorithms for parsing and finding topic keywords.

To conclude this evaluation, I present in Figure 4.2 a sample of headlines produced by the WIDL-based system, which includes both good and not-so-good outputs.

4.4 Conclusions

In this chapter, I have presented a headline-generation application based on WIDL-expressions. This application takes as input text documents, and creates, from bits and pieces of textual information, WIDL-expressions that compactly represent many possible headline realizations. The WIDL-expressions are intersected with language models, and the best scoring realizations are presented as the output headlines.

This abstractive, bottom-up approach is compared and contrasted with extractive approaches, which build their output starting from an existing sentence in the document. The evaluation carried in this chapter shows that the WIDL-based headline generation system performs at the same level of accuracy, perhaps slightly better, than an extractive, state-of-the-art system. At the same time, it outperforms a previously-proposed abstractive approach by a wide margin.

Chapter 5

Machine Translation using WIDL-expressions

5.1 Previous approaches to Statistical Machine Translation

Statistical approaches to Machine Translation are currently recognized as the most successful over the last few years in the international machine translation competition (DARPA TIDES program, Machine Translation Evaluation on Chinese-English and Arabic-English, 2003–2005). The field of Statistical Machine Translation (SMT) started in the late 1980's with IBM's Candide project (Brown *et al.* 1993), with a proposal of a translation model mapping words into words and allowing for deletion, insertion, and movement of words.

Moving away from word-level models, more modern SMT systems define translation models at phrase-level (Och, Tillmann, & Ney 1999; Och & Ney 2002; Koehn, Och, & Marcu 2003), and use a log-linear combination of feature functions capturing various levels of translation knowledge to achieve increased levels of performance. The linear combination coefficients are trained using a discriminative training technique (Och 2003) which maximizes the performance of the overall model for a specific utility function such as BLEU (Papineni *et al.* 2002).

These phrase-based systems scale well with large amounts of training date, and have provided the best performance in the aforementioned international machine translation competition.

Additionally, there is an increased effort to build translation models that use either linguistically-motivated trees, automatically generated by a syntactic parser (Yamada & Knight 2001; Schafer & Yarowsky 2003; Galley *et al.* 2004) or syntactic reordering patterns (Wu 1997; Alshawi, Bangalore, & Douglas 2000; Chiang 2005).

In this chapter, I show how a WIDL-based approach to machine translation can capture the same translation knowledge as a phrase-based translation system, and achieve translation accuracy levels comparable with a state-of-the-art translation system (Koehn 2004).

5.2 Automatic Creation of MT-specific WIDL-expressions

The machine translation system I describe in this chapter is built using the architecture presented in Figure 1.1 (Chapter 1). The generation engine implements the generation algorithms described in Chapter 3. In this section, I describe the algorithm used to implement an MT-specific WIDL-expression creation module, which automatically creates WIDL-expressions starting from foreign-language sentences. In the experiments reported here, the foreign language is taken to be Chinese, and the target language is taken to be English.

5.2.1 WIDL-expressions for Multiple Probability Distributions

The accuracy of a translation that uses WIDL-expressions as "meaning representation" is strongly correlated with translation knowledge these expressions incorporate, that is, both the coverage

of the possible ways to express a translation and the weighting scheme associated with these translations. As state-of-the-art approaches to SMT suggest, incorporating multiple translation models – such as $p(\overline{f}|\overline{e})$ (probability of foreign phrase given English phrase), $p(\overline{e}|\overline{f})$ (probability of English phrase given foreign phrase), $p_{\text{lex}}(\overline{f}|\overline{e})$ (lexical-level probability of foreign phrase given English phrase), and $p_{\text{lex}}(\overline{e}|\overline{f})$ (lexical-level probability of English phrase given foreign phrase) – as translation knowledge is beneficial (Koehn, Och, & Marcu 2003), as each of these models may capture different aspects of translation variation. At the same time, the integration of these multiple translation models with other stochastic models used in translation, such as n-gram language models, is done in a log-linear fashion, with the linear coefficients set by a discriminative training procedure which considers all these translation-knowledge sources at once (Och 2003).

The WIDL formalism can be extended in a straightforward manner to accommodate the need to handle multiple probabilistic input-biases (in this case, translation models) at once. For a number $T \geq 1$, such an extension uses the same syntax for the operators $\cdot$ (precedence), $\|$ (interleave) and $\times$ (lock) operators as described in Section 2.1, while the $\vee$ operator is specified as follows:

T-Way Weighted Disjunction. If $\omega_1, \omega_2, \ldots, \omega_n$ are WIDL-expressions (in the extended formalism), then $\omega = \vee_{\delta_0}(\omega_1, \omega_2, \ldots, \omega_n)$, with $\delta_0 : \{1, \ldots, n\} \rightarrow ([0,1])^T$ specified such that $\Sigma_{a \in \mathsf{dom}(\delta_0)} \mathsf{proj}_j(\delta_0(a)) = 1$, $1 \leq j \leq T$, is a WIDL-expression. The semantic mapping $\sigma_{\text{widl}}(\omega)$ represents T distinct probability distribution δ^j, $1 \leq j \leq T$, over the set of all strings

encoded by its argument expressions, induced by δ_0 and the distributions $\sigma_{\text{widl}}(\omega_i)$, $1 \leq i \leq n$. An example of such extended WIDL-expression is given in Figure 5.1, for $T = 4$.

One should note that, even though the syntax of the other three operators does not change, their interpretation does change, as the overall interpretation of these extended WIDL-expressions is in terms of T independent probability distributions over the finite set encoded.

5.2.2 Automatic Creation of WIDL-expressions

The module that implements the algorithm used to create MT-specific WIDL-expressions uses a probabilistic dictionary, represented as a phrase-based translation table. This probabilistic dictionary is extracted, in an unsupervised manner, from a parallel corpus of sentence-level aligned Chinese-English translations (Koehn, Och, & Marcu 2003). As mentioned already, these phrase-level translation pairs $(\overline{f}, \overline{e})$ are learnt together with four translation model probability distributions: $p(\overline{f}|\overline{e}), p(\overline{e}|\overline{f}), p_{lex}(\overline{f}|\overline{e})$, and $p_{lex}(\overline{e}|\overline{f})$, where $\overline{f}$ and $\overline{e}$ are Chinese and English phrases, respectively.

I use an algorithm resembling probabilistic bottom-up parsing to build a WIDL-expression for an input Chinese string: each contiguous span (i, j) over a Chinese string $C_{i,j}$ is considered a possible "constituent", and the "non-terminals" associated with each constituent are the English phrase translations $E^k_{i,j}$ that correspond in the translation table to the Chinese string $C_{i,j}$. Multiple-word English phrases, such as $\mathsf{w_1\ w_2\ w_3}$, are represented as WIDL-expressions using the precedence $(\cdot)$ and lock $(\times)$ operators, as $\times(\mathsf{w_1} \cdot \mathsf{w_2} \cdot \mathsf{w_3})$. To limit the number of possible translations $E^k_{i,j}$ corresponding to a Chinese span $C_{i,j}$, I use a probabilistic beam b and

a histogram beam s to beam out low probability translation alternatives. At this point, each $C_{i,j}$ span is "tiled" with likely translations $E^k_{i,j}$ taken from the translation table, or, if the span length is 1 and no translation is available, by a single "translation" that copies over the Chinese span.

Tiles that are adjacent are joined together in a larger tile by a $\|_\delta$ operator, where $\delta = \{\mathsf{perms} \overset{\mathsf{exp\_mov\_pen}}{\rightarrow} 1\}$. That is, reordering of the component tiles are permitted by the $\|_\delta$ operators (assigned non-zero probability), but the longer the movement from the original order of the tiles, the lower the probability. However, such movements can occur with enough support from other probabilistic sources, such as the language models. When multiple tiles are available for the same span (i, j), they are joined by a $\vee_\delta$ operator. Each instance of a $\vee_\delta$ operator is written as a 4-way weighted disjunction (Section 5.2.1). The entries in the 4-way distribution function δ specification are taken from the corresponding probability distribution in the translation table, if the span $C_{i,j}$ has a translation $E^k_{i,j}$ in the translation table, or is set to 1, if the tile is a $\|$-composition of smaller tiles.

As a result of this bottom-up algorithm, the input Chinese string is hierarchically tiled with English translations for contiguous spans of the Chinese string. The WIDL-expression in Figure 5.1(b) is an example of expression created by this algorithm.

A WIDL-expression created by this algorithm, using an average of $m = 38$ tiles per sentence (for an average input sentence length of 30 words) and an average of $k = 9$ possible translations per tile (for appropriate setting of the b and s beam values) encodes a candidate set that has, on average, about 10^{50} possible translations. For long input sentences, the size of the candidate set can be as high as 10^{100} possible translations. As the specification of the $\|_\delta$

(a) Input:

枪手 被 警方 击毙 .

(b) WIDL-expression:

$$\|_{\delta_1} \; (\vee_{\delta_2}(\mathsf{gunman}, \times(\mathsf{the} \cdot \mathsf{gunmen}), \mathsf{gunmen}),$$
$$\vee_{\delta_3}(\times(\mathsf{have} \cdot \mathsf{been}), \mathsf{being}, \times(\mathsf{had} \cdot \mathsf{been}), \mathsf{were}, \mathsf{was}),$$
$$\times(\mathsf{by} \cdot \mathsf{police}), \vee_{\delta_4}(\mathsf{kill}, \mathsf{killed}, \mathsf{killing}), . \;)$$

$$\delta_1 = \{\mathsf{perms} \overset{\text{exp\_mov\_pen}}{\rightarrow} 1.0\} \qquad \delta_2 = \{1 \rightarrow 0.35\ 0.03\ 0.19\ 0.25,$$
$$\delta_3 = \{1 \rightarrow 0.13\ 0.14\ 0.18\ 0.07, \qquad 2 \rightarrow 0.35\ 0.10\ 0.32\ 0.36,$$
$$2 \rightarrow 0.47\ 0.15\ 0.30\ 0.46, \qquad 3 \rightarrow 0.30\ 0.87\ 0.49\ 0.39\}$$
$$3 \rightarrow 0.14\ 0.11\ 0.18\ 0.07, \quad \delta_4 = \{1 \rightarrow 0.65\ 0.33\ 0.25\ 0.63,$$
$$4 \rightarrow 0.16\ 0.27\ 0.24\ 0.24, \qquad 2 \rightarrow 0.23\ 0.50\ 0.53\ 0.26,$$
$$5 \rightarrow 0.10\ 0.33\ 0.10\ 0.16\} \qquad 3 \rightarrow 0.12\ 0.16\ 0.22\ 0.11\}$$

(c) Output:

gunman was killed by police .

Figure 5.1: From an input foreign-language string (a), a WIDL-expression (b) is automatically created, representing four different probability distributions. By intersecting these distributions with a trigram language model, word-count and phrase-count models, the output (c) is obtained.

operators takes space $\mathrm{O}(1)$, Theorem 2 (Chapter 2) guarantees that these WIDL-expressions encode compactly these huge spaces in $\mathrm{O}(mk)$.

In the generation phase, I use the WIDL-based generation engine to interpolate the distribution probabilities of WIDL-expressions with stochastic language models. In the notation used for Equation 3.2, I use four feature functions $h_0, \ldots, h_3$ for the WIDL-expression distributions (one for each probability distribution encoded). I also use a feature function h_4 for a trigram language model; a feature function h_5 for a word-count model (which counts the number of words in a proposed realization); and a feature function h_6 for a phrase-count model (which counts the number of phrases in a proposed realization). These count models allow the

log-linear model to learn, when properly trained, to produce translations that have length proportional with the length of the translated input, by preferably using few phrases (as few phrases means long phrases, which tend to create a more fluent output).

As acknowledged in the Machine Translation literature (Germann *et al.* 2001), full A* search is not usually possible, due to the large size of the search spaces. I therefore use a faster – but less accurate – version of the WIDL-NGLM-A* algorithm, called the k-steps version (Chapter 3, Section 3.4). The notation I use for this algorithm is WIDL-NGLM-A$^*_k$, and the experiments reported here are run with $k = 2$.

As an example, consider the Chinese sentence[1] in Figure 5.1. The $\vee_\delta$ operators specify four probability values, one for each translation distribution in the translation table. The $\|_{\delta_1}$ operator specifies as interleave distribution the exponential movement penalty model (Chapter 2, Section 2.1). The best scoring translation found by the WIDL-NGLM-A* algorithm, in conjunction with a trigram, word-count, and phrase-count model is gunman was killed by police .

5.3 Machine Translation Evaluation

I evaluate the performance of the WIDL-based Chinese-English machine translation application using the 2003 NIST MT evaluation set for testing and BLEU (Papineni *et al.* 2002) as the evaluation metric. The comparison is made against a publicly available, state-of-the-art phrase-based decoder, Pharaoh (Koehn 2004). The resources used by Pharaoh and the WIDL-based algorithm are the same: a translation table trained on the FBIS corpus (7.2M Chinese

[1] English translation: the gunman was shot dead by the police.

words and 9.2M English words of parallel text), and a trigram language model trained on 155M words of English newswire using the SRI Language Model Toolkit (with modified Kneser-Ney smoothing). The algorithm that creates WIDL-expressions uses the probabilistic beam b set to 0.01, and the histogram beam s set to 10. Running Pharaoh with its equivalent beams set to the same values, probabilistic beam b set to 0.01, and maximum size of beam s set to 10, provides a meaningful comparison between the two systems. However, these values are more restrictive than the default values for the beams used by Pharaoh (the probabilistic beam b set to 0.00001, the histogram beam s set to 100), and therefore I also measure Pharaoh's performance using its default values. The interpolation weights $\overline{\lambda}$ (Equation 3.2) are trained using discriminative training (Och 2003) using BLEU as the objective function, on the 2002 NIST MT evaluation set.

The results of the evaluation are presented in Table 5.1. The BLEU score for Pharaoh using its default beam values is 0.2680, whereas its performance using the more restrictive beam values is 0.2635. The BLEU score for the WIDL-based system, using the restrictive beam values, is 0.2570. The difference in score between these two comparable runs is not statistically significant at 95% confidence (using bootstrap resampling). These results show that the WIDL-based approach to machine translation is powerful enough to achieve translation accuracy comparable with state-of-the-art systems in machine translation.

MT system	Beam settings	BLEU n=4	n-gram precision 1	2	3	4
Pharaoh	$s = 100$, $b = 10^{-5}$	0.2680	0.72	0.37	0.19	0.10
Pharaoh	$s = 10$, $b = 10^{-2}$	0.2635	0.72	0.36	0.19	0.10
WIDL-NGLM-A$_2^*$	$s = 10$, $b = 10^{-2}$	0.2570	0.72	0.36	0.18	0.09

Table 5.1: Machine translation evaluation. The performance of state-of-the-art Pharaoh system and WIDL-based system is measured using BLEU.

5.4 Conclusions

In this chapter, I have presented a machine translation application based on WIDL-expressions. This application takes as input foreign-language sentences, and creates, using a probabilistic, phrase-based dictionary, WIDL-expressions that compactly represent many possible translations. The WIDL-expressions are intersected with language models, and the best scoring realizations are presented as the output translations.

The evaluation carried in this chapter shows that the WIDL-based translation system performs at levels of accuracy that are comparable, albeit slightly worse, with the performance of a state-of-the-art translation system. Previous attempts to use generation-oriented approaches to machine translation have achieved a much less impressive performance, below the performance of a word-for-word statistical translation system (Hajic *et al.* 2002; Habash 2003).

Chapter 6

WIDL-expressions for Generating Coherent Discourse

6.1 Previous Approaches to Discourse Coherence

Various theories of discourse coherence (Mann & Thompson 1988; Grosz, Joshi, & Weinstein 1995) have been applied successfully in discourse analysis (Marcu 2000; Forbes *et al.* 2001) and discourse generation (Scott & de Souza 1990; Kibble & Power 2004). Most of these efforts, however, have limited applicability. Those that use manually written rules model only the most visible discourse constraints (e.g., the discourse connective "although" marks a CONCESSION relation), while being oblivious to fine-grained lexical indicators. The methods that utilize manually annotated corpora (Carlson, Marcu, & Okurowski 2003; Karamanis *et al.* 2004) and supervised learning algorithms require trained annotators and intensive labor to produce the annotations, which usually cannot be produced in large amounts.

In contrast, more recent research has focused on stochastic approaches that model discourse coherence at the local lexical (Lapata 2003) and global levels (Barzilay & Lee 2004), while preserving regularities recognized by classic discourse theories (Barzilay & Lapata 2005). These

stochastic coherence models use simple, non-hierarchical representations of discourse and can be trained in an unsupervised manner using large collections of human-authored documents. The unsupervised nature of these models increases their scalability and portability.

As each of these stochastic models captures different aspects of coherence, an important question is whether they can be combined in a model capable of exploiting all coherence indicators. In Chapter 7, I show that a WIDL-based approach to discourse coherence provides an affirmative answer to this question. In this chapter, I present previously-proposed, as well as new models of discourse coherence (Section 6.2), and also describe how to use the WIDL formalism to represent possible realizations at discourse level (Section 6.3).

6.2 Stochastic Models of Discourse Coherence

6.2.1 Local Models of Discourse Coherence

Stochastic local models of coherence work under the assumption that well-formed discourse can be characterized in terms of specific distributions of local recurring patterns. These distributions can be defined at various levels, e.g. at word level or at entity level.

Word-Coocurrence Coherence Models. I define here a new coherence model that models the intuition that the usage of certain words in a discourse unit (sentence) tends to trigger the usage of other words in subsequent discourse units. A similar intuition holds for the Machine Translation models generically known as the IBM models (Brown *et al.* 1993), which assume

that certain words in a source language sentence tend to trigger the usage of certain words in a target language translation of that sentence.

I train models able to recognize local recurring patterns of word usage across sentences in an unsupervised manner, by running an Expectation-Maximization (EM) procedure over pairs of consecutive sentences extracted from a large collection of training documents[1]. The motivation is that EM will be able to detect and assign higher probabilities to recurring word patterns compared to casually occurring word patterns.

A local coherence model based on IBM Model 1 assigns the following probability to a text T consisting of n sentences $s_1 s_2 \ldots s_n$:

$$P_{\mathrm{IBM}_1^{\mathrm{D}}}(T) = \Sigma_{i=1}^{n-1} \Pi_{j=1}^{|s_{i+1}|} \frac{\epsilon}{|s_i|+1} \Sigma_{k=0}^{|s_i|} t(s_{i+1}^j | s_i^k) \tag{6.1}$$

I call the above equation the direct IBM Model 1, as this model considers the words in sentence s_{i+1} (the s_{i+1}^j events) as being generated by the words in sentence s_i (the s_i^k events, which include the special s_i^0 event called the NULL word). I also define a local coherence inverse IBM Model 1:

$$P_{\mathrm{IBM}_1^{\mathrm{I}}}(T) = \Sigma_{i=1}^{n-1} \Pi_{k=1}^{|s_i|} \frac{\epsilon}{|s_{i+1}|+1} \Sigma_{j=0}^{|s_{i+1}|} t(s_i^k | s_{i+1}^j) \tag{6.2}$$

This model considers the words in sentence s_i (the s_i^k events) as being generated by the words in sentence s_{i+1} (the s_{i+1}^j events, which include the special s_{i+1}^0 event called the NULL word).

[1] I use for training the publicly-available GIZA++ toolkit, http://www.fjoch.com/GIZA++.html

Entity-based Coherence Models. Barzilay and Lapata (2005) recently proposed an entity-based coherence model that aims to learn abstract coherence properties, similar to those stipulated by Centering Theory (Grosz, Joshi, & Weinstein 1995). Their model learns distribution patterns for transitions between discourse entities that are abstracted into their syntactic roles – subject (**S**), object (**O**), other (**X**), missing (**-**). The feature values are computed using an entity-grid representation for the discourse that records the syntactic role of each entity as it appears in each sentence. Also, salient entities are differentiated from casually occurring entities, based on the widely used assumption that occurrence frequency correlates with discourse prominence (Morris & Hirst 1991; Grosz, Joshi, & Weinstein 1995).

The probability assigned to a text $T = s_1 \dots s_n$ by this Entity-Based model (henceforth called EB) can be written as:

$$P_{\mathrm{EB}}(T) = \Sigma_{i=1}^{n-1} \Sigma_{k=1}^{M} w_k f_k(s_{i+1}|s_i) \tag{6.3}$$

Here, $f_k(s_i, s_{i+1})$ are feature values, and w_k are weights trained to discriminate between coherent, human-authored documents and examples assumed to have lost some degree of coherence (scrambled versions of the original documents).

6.2.2 Global Models of Discourse Coherence

Barzilay and Lee (2004) propose a document content model that uses a Hidden Markov Model (HMM) to capture more global aspects of coherence. Each state in their HMM corresponds

to a distinct "topic". Topics are determined by an unsupervised algorithm via complete-link clustering.

The probability assigned to a text $T = s_1 \ldots s_n$ by this Content Model (henceforth called CM) can be written as:

$$P_{\mathrm{CM}}(T) = \max_{t_1 \ldots t_n} \Pi_{i=1}^{n} P_{\mathrm{TT}}(t_i|t_{i-1}) \times P_{\mathrm{TM}}(s_i|t_i) \tag{6.4}$$

The first term, P_{TT}, models the probability of changing the topic as a bigram model over topics. The second term, P_{TM}, models the probability of generating sentences from topic t_i, as a bigram language model conditioned on topic t_i.

6.2.3 Combining Local and Global Models of Discourse Coherence

I define a new model of discourse coherence using a log-linear model that combines all the discourse coherence models presented above. By expressing these coherence models as a set of M feature functions $h_m(T)$, $1 \leq m \leq M$, and using a model parameter λ_m for each feature function, One can formulate the search problem of finding the most probable text T under a log-linear model as in Equation 3.2 (Chapter 3, Section 3.2), which in the context of coherence modeling can be written as Equation 6.5:

$$\arg\max_T P(T) = \arg\max_T \exp[\Sigma_{m=1}^{M} \lambda_m h_m(T)] \tag{6.5}$$

$$\arg\max_T P(T) = \arg\min_T -\Sigma_{m=1}^{M} \lambda_m \log h_m(T)$$

(a)

A: It said no information had been received about injuries or damage from the magnitude 6.1 quake which struck the sparsely inhabited area at 2 43 AM (1843 GMT)

B: BC−China−Earthquake|Urgent Earthquake rocks northwestern Xinjiang Mountains

C: Beijing (AP) A strong earthquake hit the Altai Mountains in northwestern Xinjiang early Wednesday the official Xinhua News Agency reported

D: (EK)

(b)

It	information	injuries	damage	magnitude	quake	area	GMT	BC−China...	Earthquake	northwestern	Xinjiang	Mountains	Beijing	AP	Altai	Wednesday	Xinhua	News	Agency	EK
S	S	X	X	X	X	O	X	–	–	–	–	–	–	–	–	–	–	–	–	–
–	–	–	–	–	–	–	–	S	S	O	O	O	–	–	–	–	–	–	–	–
–	–	–	–	–	–	–	–	–	S	X	X	O	S	X	O	X	S	S	S	–
–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	X

(c)

α: "it said no information had been received about injuries or damage from the magnitude +.+ quake which struck the sparsely inhabited area at + ++ am (++++ gmt) ## SSXXXXOX−−−−−−−−−−−−−"

β: "−Name− earthquake rocks northwestern −Name− −Name− ## −−−−−−−−SSOOO−−−−−−−−"

γ: "−Name− (−Name−) a strong earthquake hit the −Name− −Name− in northwestern −Name− early −Name− the official −Name− −Name− −Name− reported ## −−−−−−−−−SXXOSXOXSSS−"

δ: "(ek) ## −−−−−−−−−−−−−−−−−−−X"

Figure 6.1: An example consisting of four discourse units (A, B, C, and D) is presented in (a). In (b), their entities are detected (underlined) and assigned syntactic roles: **S** (subject), **O** (object), **X** (other), **-** (missing). In (c), terms α, β, γ, and δ encode the discourse units properties of these units for model scoring purposes.

In this framework, I distinguish between the *modeling* problem, which amounts to finding appropriate feature functions for the discourse coherence task, and the *training* problem, which amounts to finding appropriate values for $\lambda_m, 1 \leq m \leq M$. I address the modeling problem by using as feature functions the discourse coherence models presented in the previous sections, namely model IBM_1^{D} (Equation 6.1), model IBM_1^{I} (Equation 6.2), model EB (Equation 6.3), and models TT and TM (Equation 6.4). In the next chapter, I will address the training problem by performing a discriminative training procedure of the λ_m parameters, using as utility functions different metrics used to measure discourse coherence.

6.3 Discourse-level WIDL-expressions

In this section, I describe how to use the WIDL formalism to represent possible realizations at the level of discourse, that is, above sentence level. Without losing generality, I will consider sentences as discourse units in our examples and experiments. Consider the discourse units A-D presented in Figure 6.1(a). Each of these units undergoes various processing stages in order to provide the information needed by the coherence models needed by the log-linear model involved in Equation 6.5. The entity-based model (EB) (Section 6.2), for instance, makes use of a syntactic parser (Collins 2003) to determine the syntactic role played by each detected entity (Figure 6.1(b)). For example, the string SSXXXXOX---------- (first row of the grid in Figure 6.1(b), corresponding to discourse unit A) encodes that It and information have subject (S) role, injuries, etc. have other (X) roles, area has object (O) role, and the rest of the entities do not appear (-) in this unit.

In order to be able to compute a solution to Equation 6.5, the input representation needs to provide the necessary information to compute all h_m terms, that is, all individual model scores. Textual units A, B, C, and D in our example are therefore represented as atomic WIDL-expressions α, β, γ, and δ, respectively[2] (Figure 6.1(c)). Given M individual models, each defined over an alphabet Σ_i, $1 \leq i \leq M$, atomic WIDL-expressions are defined over a finite input alphabet defined as $\Sigma = \Pi_{i=1}^{M} \Sigma^i$ and a distribution function alphabet Δ.

For the coherence models considered in this chapter, given that models IBM_1^{D} (Equation 6.1), IBM_1^{I} (Equation 6.2), and CM (Equation 6.4) (as far as the observed events are concerned) are

[2]Following Barzilay and Lee (2004), proper names, dates, and numbers are replaced with generic tokens.

all lexicalized models, they may be considered as being defined using a common vocabulary Voc as strings in Voc^*. On the other hand, the EB model (Equation 6.3) is non-lexicalized, defined using strings from $(\{\mathbf{S}, \mathbf{X}, \mathbf{O}, -\})^*$. In this case, the atomic WIDL-expressions α, β, γ, and δ are defined over an input alphabet $\Sigma \stackrel{\text{def}}{=} \mathsf{Voc}^* \times (\{\mathbf{S}, \mathbf{X}, \mathbf{O}, -\})^*$ (in Figure 6.1(c), the symbol $\sharp\sharp$ is used to separate the two sides of this Cartesian product) and a distribution function alphabet Δ.

6.3.1 WIDL-expressions for Information Ordering

Consider now a first example WIDL-expression encoding different discourse orderings,

$$E_1 = \langle d \rangle \cdot \|_{\delta_0}(\alpha, \beta, \gamma, \delta) \cdot \langle /d \rangle \quad \delta_0 = \{\mathsf{perms} \stackrel{\mathsf{uniform}}{\rightarrow} 1\}$$

E_1 uses the $\|_{\delta_0}$ operator to create a uniform distribution over a bag of units. That is, E_1 stands for all possible order permutations, equally weighted, of α, β, γ, and δ, with the additional information that any of these orders are to appear between the beginning $\langle d \rangle$ and end of document $\langle /d \rangle$. Such an expression is useful in an information ordering scenario, in which the coherence models are employed to select an order for the input units that makes for the most coherent discourse, in the absence of any other information relevant to ordering.

6.3.2 WIDL-expressions for Multi-Document Summarization

Another possible scenario in which E_1 can be used is multi-document summarization, in which a summarization component decides that units A, B, C, and D represent the most important units

of information to be presented, and creates expression E_1 to be passed to a subsequent module in charge with selecting an order that makes for the most coherent summary. In this scenario, however, the summarization component may also know, for instance, that unit A occurs right before D in some original document. Therefore, it may want to forbid any insertion of extra-material in between A and D, so that their coherence is preserved. In this case, it may create expression

$$E_2 = \langle d \rangle \cdot \|_{\delta_0}(\times(\alpha, \delta), \beta, \gamma) \cdot \langle /d \rangle \quad \delta_0 = \{\mathsf{perms} \overset{\mathsf{uniform}}{\rightarrow} 1\}$$

which enforces this interpretation using the $\times$ operator over α and δ. A more sophisticated summarization module may also decide to create or consider additional renderings of, say, units B (with encodings β, β', β'') and C (with encodings γ, γ'), to allow for more coherent outcomes. These different renderings may receive different weights, reflecting, for instance, the amount of information content captured by each different rendering. In this case, expression

$$E_3 = \langle d \rangle \cdot \|_{\delta_0}(\times(\alpha, \delta), \vee_{\delta_1}(\beta, \beta', \beta''), \vee_{\delta_2}(\gamma, \gamma')) \cdot \langle /d \rangle$$

$$\delta_0 = \{\mathsf{perms} \overset{\mathsf{uniform}}{\rightarrow} 1\}$$

$$\delta_1 = \{1 \rightarrow p_1, 2 \rightarrow p_2, 3 \rightarrow p_3\} \quad \delta_2 = \{1 \rightarrow q_1, 2 \rightarrow q_2\}$$

encodes all these possible alternatives using the weighted $\vee$ operator. As the goal is to achieve both increased information content and increased coherence in the final summary, the input-bias

introduced by the $\vee_{\delta_i}$ operators can be tightly integrated with the language model bias introduced by the discourse coherence models, such that the best realization is both an informative and coherent summary.

6.4 Conclusions

In this chapter, I have presented several stochastic models of discourse coherence, which use different means to capture aspects that characterize text coherence. I have also presented the WIDL formalism from a discourse-level perspective, and I have described possible alternatives to creating WIDL-expressions that can be used, in conjunction with discourse coherence models, to generate coherent discourse.

In the next chapter, I present the algorithms that perform the intersection of discourse-level WIDL-expressions with stochastic discourse coherence models. These algorithms are also crucial for training the parameters of the log-linear model of coherence introduced in this chapter.

Chapter 7

WIDL-based Coherent Discourse Computation and Utility-Based Training

A frequently used testbed for coherence models is the discourse ordering problem, which occurs often in text generation, complex question answering, and multi-document summarization: given N discourse units, what is the most coherent ordering of them (Lapata 2003; Barzilay & Lee 2004; Barzilay & Lapata 2005)? Because the problem is NP-complete (Althaus, Karamanis, & Koller 2005), it is critical how coherence model evaluation is intertwined with search: if the search for the best ordering is greedy and has many errors, one is not able to properly evaluate whether a model is better than another. If the search is exhaustive, the ordering procedure may take too long to be useful.

In this chapter, I show how a WIDL-based A* search algorithm that comes with strong theoretical guarantees can be applied to the discourse ordering problem . For a wide range of practical problems (discourse ordering of up to 15 units), the algorithm finds the *optimal* solution in reasonable time (on the order of seconds). A beam search version of the algorithm enables

one to find good, approximate solutions for very large reordering tasks. These algorithms enable one not only to compare head-to-head, for the first time, a set of coherence models, but also to combine these models so as to benefit from their complementary strengths. The model combination is accomplished using statistically well-founded utility training procedures which automatically optimize the contributions of the individual models on a development corpus. I empirically show that utility-based models of discourse coherence outperform each of the individual coherence models considered.

7.1 Algorithms for Intersecting WIDL-expressions with Stochastic Coherence Models

The algorithms presented in Chapter 3 for intersecting WIDL-expressions with n-gram language models can be extended to perform the intersection of discourse-level WIDL-expressions (Section 6.3) with stochastic models of discourse coherence (Section 6.2).

First, I make the observation that the word-coocurrence coherence models IBM_1^{D} and IBM_1^{I} from Equations 6.1 and 6.2, as well as the entity-based model EB from Equation 6.3 (Chapter 6) are local models, based on a markovian assumption that the probability of a certain event depends only on the previous n surface events, where $n = 1$ in these particular cases. It follows that, similar with the formulation for bigram language models (which also uses a markovian assumption with history length $n = 1$), relation UNFOLD_ω^1 (see Chapter 2, Section 2.1) is defined in a similar manner. More concretely, for a discourse-level WIDL-expression ω defined over Σ and Δ, a relation $(C, X, C') \in \text{UNFOLD}_\omega^1$ consists of states C, $C' \in 1\text{-}\mathsf{cut}(\omega)$, with

$X \in \Sigma$ and state $C = (c, h, s)$, where c represents a set of vertices in the WIDL-graph γ_ω which can be reached simultaneously while unfolding the graph, h is a 1-history string, $h \in \Sigma$, and $s \in (\mathbf{N} \cup \{\langle_\delta, \rangle_\delta | \delta \in \Delta\})^*$ is a $\|$-stack string.

7.1.1 Intersecting WIDL-expressions with Hidden-variable Coherence Models

Model CM (Equation 6.4) uses hidden variables (topics t_i) to compute the probability it assigns to a given text T. Assuming the hidden variables to come from a finite alphabet H, one can formulate algorithms that operate in a similar regime with the WIDL-NGLM-A* and WIDL-NGLM-BEAM algorithms (Chapter 3), but are capable of handling models with hidden variables of the type presented in Equation 6.4.

Algorithm WIDL-CH-A* This algorithm uses the A* searching technique on a model that results from the log-linear combination of a WIDL-graph G_ω probability distribution and M discourse coherence models (Equation 6.5), using $\overline{\lambda}$ as coefficients. This algorithm is presented in Figure 7.1.

Algorithm WIDL-CH-A* computes relation UNFOLD$^1_\gamma$ for G_ω (see Chapter 2) incrementally, using a set of active states, called *active* (line 4). For each hidden variable $i \in H$, it creates a separate copy of each element in UNFOLD$^1_\gamma$, which is used to EVALUATE the cost of the current state using Equation 6.5 (line 5). EVALUATE also uses an admissible heuristic function (Russell & Norvig 1995) to compute future (admissible) costs for the *unfold* states in the presence of hidden variables, as defined in the next section. The unfolding process is controlled using a priority queue Q that sorts (from min to max) the states according to their total cost

```
WIDL-CH-A*(G_ω, CH̄, λ̄)
1  active ← {[vs^{G_ω}]} j ← 0
2  flag ← 1 Q ← ∅
3  while flag
       do unfold ← ∅
          for each i in H
4              do unfold_i ← UNFOLD(active, G_ω)
5                 EVALUATE(unfold_i, i, j, CH̄, λ̄)
6                 if FINAL(unfold_i, G_ω)
                    then flag ← 0
                  unfold ← unfold ∪ ⟨unfold_i, i⟩
7         for each state in unfold
              do PUSH(Q, state)
          ⟨active, j⟩ ← POP(Q)
8  return active
```

Figure 7.1: Pseudo-code for intersecting a WIDL-graph G_ω with discourse coherence models $\overline{CH}$ with hidden variables using incremental unfolding and A* search.

(current + admissible). In the next iteration, *active* is a singleton set containing the state POPed out from the top of Q (the lowest cost state), which remembers the hidden variable j used to compute its score. Therefore, it can be used by the EVALUATE function as a history context of length 1.

Algorithm WIDL-CH-BEAM If, instead of controlling the unfolding using a priority queue, I control the unfolding using a value *beam* and a BEAMSTATES function, I obtain a beam search algorithm which operates under models with hidden variables (Figure 7.2). The beam search comes in two flavors, namely probabilistic beam and histogram beam search (see Chapter 3, Section 3.4).

```
WIDL-CH-BEAM(G_ω, CH̄, λ̄, beam)
1  active ← {[vs^{G_ω}]} j̄ ← 0̄
2  flag ← 1 B ← ∅
3  while flag
4       do unfold ← ∅
           for each i in H
5               do unfold_i ← UNFOLD(active, G_ω)
6                  EVALUATE(unfold_i, i, j̄, CH̄, λ̄)
7                  if FINAL(unfold_i[0], G_ω)
                     then flag ← 0
                   unfold ← unfold ∪ ⟨unfold_i, i⟩
8          ⟨active, j̄⟩ ← BEAMSTATES(unfold, beam)
           for each ⟨state, j⟩ in ⟨active, j̄⟩
               do if FINAL(state, G_ω)
                     then PUSH(B, ⟨state, j⟩)
9  SORT(B)
   return POP(B)
```

Figure 7.2: Pseudo-code for intersecting a WIDL-graph G_ω with discourse coherence models $\overline{CH}$ with hidden variables using incremental unfolding and beam search.

Similar with WIDL-CH-A*, this algorithm creates a separate copy, for each hidden variable $i \in H$, of each element in UNFOLD$^1_\gamma$. Each copy, together with its hidden label, is used to EVALUATE the cost of the current state using Equation 6.5. EVALUATE also uses an heuristic function to compute future (admissible) costs for the *unfold* states in the presence of hidden labels (next section), such that the BEAMSTATES function keeps only the most promising states – computed using current + admissible cost – in the set *active*, together with their respective hidden labels (vector $\overline{j}$). The vector of hidden labels $\overline{j}$ is used in the next iteration by the EVALUATE function as a history context of length 1 to compute the cost of each newly unfold state under hidden label i. The unfolding stops when the first state in the *unfold* set is also final.

At this point, the algorithm sorts the queue B of final states and returns the first state in the queue, i.e., the one with the lowest cost.

7.1.2 Computing Admissible Heuristics for WIDL-expressions Intersected with Hidden-Variable Models

To compute admissible heuristics for WIDL-expressions intersected with models with hidden variables such as model CM (Equation 6.4), I use a technique similar with the one described in Chapter 3, Section 3.5.

For each state S in a probabilistic finite-state acceptor A_ω^1 (the superscript 1 stands for the 1-history strings recorded by the pFSA states) corresponding to a WIDL-graph G_ω, one can efficiently extract from G_ω – without further unfolding – the set[1] of all edge labels in Σ that can be used to reach a final state in A_ω^1. This set of labels, denoted FE_S^{all}, is an overestimation of the set of future events reachable from S, because the labels under the $\vee$ operators are all considered (whereas normally the ones used by different arguments of a $\vee$ operator are mutually exclusive). From FE_S^{all} and the 1-history string recorded in state S, I obtain a set of labels denoted FCE_S, which is an (over)estimated set of the possible future *conditioning* events for state S using the regular (visible) labels from Σ. If the alphabet of hidden labels is H, then the set $FCE_S \times H$ is an (over)estimated set of possible future conditioning events (using both visible and hidden labels) for state S, guaranteed to contain the most cost-efficient conditioning events in the future of S.

[1] Actually, these are multi-sets, as I treat multiply-occurring labels as separate items.

Using FCE_S, one needs to extract from FE_S^{all} the set of most cost-efficient future *events* from under each $\vee$ operator, denoted FE_S. Then the set $FE_S \times H$ is the set of all future events (using both visible and hidden labels) for state S. The sets $FE_S \times H$ and $FCE_S \times H$ are used to formulate an admissible heuristic cost for state S under a log-linear combination of M coherence models $\overline{CH}$ containing hidden variables employed as in Equation 6.4, using Equation 7.1:

$$h_\omega(S) = \Sigma_{e \in FE_S} \Sigma_{m=1}^{M} \lambda_m \min_{\substack{i \in H \\ \langle ce, j \rangle \in (FCE_S \times H)}} -\log p_{CH\,m}(\langle e, i \rangle | \langle ce, j \rangle) \tag{7.1}$$

If $h_\omega^*(S)$ is the true future model cost for state S, Equation 7.1 guarantees that $h_\omega(S) \leq h_\omega^*(S)$ from the way the set of future events $FE_S \times H$ and the set of future conditioning events $FCE_S \times H$ are constructed.

7.1.3 Formal Properties of WIDL-CH Algorithms

The following theorem states the correctness of the proposed WIDL-CH algorithms. Algorithm WIDL-CH-A* comes with the strong guarantee that it finds the maximum probability path encoded by a WIDL-graph under the log-linear combination specified by Equation 6.5. Algorithm WIDL-CH-BEAM comes with a weaker guarantee of finding the maximum probability path, but allows for a faster search while maintaining good approximations for the best path.

Theorem 7 *Let ω be a WIDL-expression, G_ω its WIDL-graph, and A_ω^1 its corresponding pFSA recording 1-history strings. Let $\overline{h} = \langle G_\omega, CH_1 \ldots CH_M \rangle$ (CH_i a discourse coherence model*

of the type presented in Equations 6.1-6.4, $1 \leq i \leq M$*, with* H *the set of hidden variables),* $\overline{\lambda} = \langle \lambda_0, \lambda_1, \ldots, \lambda_M \rangle$ *be a vector of real numbers, and* b *also a real number. Algorithm* WIDL-CH-A$^*(G_\omega, \overline{CH}, \overline{\lambda})$ *finds the path of maximum probability under Equation 6.5. Algorithm* IDL-CH-BEAM$(G_\omega, \overline{CH}, \overline{\lambda}, b)$ *finds the path of maximum probability under Equation 6.5, if all* $\langle s, j \rangle$ *pairs (*$s \in A^1_\omega, j \in H$*) along this path are selected by its* BEAMSTATES *function when using* b *as its beam parameter.*

The proof for this theorem follows exactly the same arguments as the proof for Theorem 4, and therefore I omit it. The next theorem characterizes the run-time complexity of these algorithms, in terms of the complexity of the WIDL-graph G_ω corresponding to the input WIDL-expression ω and the size of the hidden variable set H used by the discourse coherence models employed.

Theorem 8 *Let* ω *be a WIDL-expression and* $G_\omega = (V, E, v_s, v_e, \lambda, r)$ *its WIDL-graph. Let* $k = \mathsf{width}(\omega)$*,* $K = \Sigma_i |active_i|$*, and* $h = |H|$*. Algorithm* WIDL-CH-A* *has run-time complexity* $\mathrm{O}(khK + hK \log hK)$*, and algorithm* WIDL-CH-BEAM *has run-time complexity* $\mathrm{O}(khK)$*. Moreover,* K *is characterized in terms of* G_ω *by the following tight upperbound:* $K \leq \left(\frac{|V|}{k}\right)^k$.

The proof for this theorem is similar with the proof for the complexity result of Theorem 6, given that, in the worse case, one needs to create one copy of each element in UNFOLD$^1_\omega$ for every hidden label in H. The conclusion is that the run-time behavior of these algorithms is linear in the complexity of the input WIDL-expression and the size of the set of hidden labels H employed by the coherence models.

However, a high-complexity WIDL-expression such as $\|_\delta(\alpha_1, \ldots, \alpha_n)$ leads to a run-time that is exponential in n (as $K \leq (\frac{2n}{n})^n = 2^n$). This is no surprise, as such an expression represents a bag of discourse-units, and finding the most coherent unit order is equivalent to the discourse ordering problem, known to be NP-complete (Althaus, Karamanis, & Koller 2005).

On the other hand, at the low-end of the complexity spectrum, for expressions such as $\alpha_1 \cdot \ldots \cdot \alpha_n$, the run-time of these algorithms is polynomial in n. As such expressions represent an actual order of the discourse units, running the algorithms on such expressions amounts to solving the discourse analysis problem using stochastic models of coherence. For instance, for $\overline{\lambda}$ such that $\lambda_{\mathrm{CM}} = 1$ and all other λs are 0, algorithm WIDL-CH-A* is guaranteed to find the sequence $t_1 \ldots t_n$ of topics that best characterizes the given discourse unit order under model CM (Equation 6.4).

7.2 Utility-based Training

In addition to the modeling problem, I must also address the training problem, which amounts to finding appropriate values for the λ_m parameters for Equation 6.5.

The solution I employ here is the discriminative training procedure of Och (2003). This procedure learns an optimal setting of the λ_m parameters using as optimality criterion the utility of the proposed solution. There are two necessary ingredients to implement Och's (2003) training procedure. First, it needs a search algorithm that is able to produce ranked k-best lists of the most promising candidates in a reasonably fast manner. This is achieved by modifying algorithm WIDL-CH-BEAM, such that instead of recording only the best scoring path, it records

a sorted list of size k of the best scoring paths, which is at the end used to extract the ranked k-best list of candidate realizations. The interested reader may consult (Huang & Chiang 2005) for algorithms on how to create k-best lists. In terms of speed, the current implementation of the WIDL-CH-HB^{100} algorithm decodes bag-of-units WIDL-expressions (see Chapter 6, Section 6.3) at an average speed of 75.4 sec./exp. on a 3.0 GHz CPU Linux machine, for an average input of 11.5 units per expression.

Second, the discriminative training procedure of Och (2003) needs a criterion which can automatically assess the quality of the proposed candidates. To this end, I employ two different metrics, such that I can measure the impact of using different utility functions on performance.

TAU (Kendall's τ). One of the most frequently used metrics for the automatic evaluation of document coherence is Kendall's τ (Lapata 2003; Barzilay & Lee 2004). TAU measures the minimum number of adjacent transpositions needed to transform a proposed order into a reference order. The range of the TAU metric is between -1 (the worst) to 1 (the best).

BLEU. One of the most successful metrics for judging machine-generated text is BLEU (Papineni *et al.* 2002). It simply counts the number of unigram, bigram, trigram, and four-gram matches between hypothesis and reference, and combines them using geometric mean. The values of the BLEU scores are between 0 (the worst) and 1 (the best).

A Comparison of Metrics. To compare the two metrics, I present in Table 7.1 three hypotheses, together with their scores measured against the reference string "1 2 3 4 5 6 7 8 9 10", using TAU and BLEU. Consider that the tokens represent sentence indexes in some original

Hypothesis	TAU	BLEU
A: 1 2 3 4 5 6 7 8 10 9	0.95	0.80
B: 10 2 3 4 5 6 7 8 9 1	0.24	0.80
C: 1 3 2 6 5 4 7 8 9 10	0.82	0.33

Table 7.1: Measuring accuracy of various hypotheses against the reference "1 2 3 4 5 6 7 8 9 10", using Kendall's τ (TAU) and BLEU.

text of length 10. From a text coherence perspective (especially news text), it is likely that hypothesis B is less coherent (starting with the last sentence) than hypothesis A. The TAU metric easily distinguishes between these two cases (0.95 versus 0.24). However, it also considers that hypothesis C is much better than hypothesis B (0.82 versus 0.24), which is counter-intuitive (C has six transitions which are different from the original order and where coherence-related problems can occur, while B has only two).

With BLEU, on the other hand, hypothesis B gets a BLEU score of 0.80 (only two transitions of length 2 are not right), whereas hypothesis C gets a BLEU score of 0.33 (six transitions of length 2 are not right). BLEU, however, is blind to phenomena like the one emphasized by hypothesis A versus B (swapping the last two sentences, versus swapping the first and last sentence): both get a score of 0.80.

I run two different discriminative training sessions using TAU and BLEU, and train two different sets of the λ_m parameters for Equation 6.5. The log-linear models thus obtained are called Log-linear$_{\mathsf{maxTAU}}$ and Log-linear$_{\mathsf{maxBLEU}}$, respectively.

7.3 Evaluation on Information Ordering

I evaluate empirically two different aspects of the work presented here. First, I measure the performance of the search algorithms across different models. Second, I compare the performance of each individual coherence model, and also the performance of the discriminatively trained log-linear models. I also compare the overall performance (model & decoding strategy) obtained using the WIDL framework with previously reported results.

7.3.1 Evaluation setting

The task on which I conduct the evaluation is information ordering (Lapata 2003; Barzilay & Lee 2004; Barzilay & Lapata 2005). In this task, a pre-selected set of information-bearing document units (in this case, sentences) needs to be arranged in a sequence which maximizes some specific information quality (in this case, discourse coherence). I use the information-ordering task as a means to measure the performance of the WIDL-CH algorithms and coherence models in a well-controlled setting. As described in Section 3.4, the WIDL framework can be used in applications such as multi-document summarization. In fact, Barzilay et al. (2002) formulate the multi-document summarization problem as an information ordering problem, and show that naive ordering algorithms such as majority ordering (select most frequent orders across input documents) and chronological ordering (order facts according to publication date) do not always yield coherent summaries.

Data. For training and testing, I use documents from two different genres: newspaper articles and accident reports written by government officials (Barzilay & Lapata 2005). The first

Algorithm	IBM_1^D		IBM_1^I		CM		EB	
	ESE	TAU BLEU	ESE	TAU BLEU	ESE	TAU BLEU	ESE	TAU BLEU
	EARTHQUAKES							
WIDL-CH-A*	0%	.39 .12	0%	.33 .13	0%	.39 .12	0%	.19 .05
WIDL-CH-B^{100}	0%	.38 .12	0%	.32 .13	0%	.39 .12	0%	.19 .06
WIDL-CH-B^{1}	4%	.37 .13	13%	.34 .14	36%	.32 .11	16%	.18 .05
Lapata, 2003	90%	.01 .04	58%	.02 .06	97%	.05 .04	46%	-.05 .00
	ACCIDENTS							
WIDL-CH-A*	0%	.41 .21	0%	.40 .21	0%	.37 .15	0%	.13 .10
WIDL-CH-B^{100}	0%	.41 .20	0%	.40 .21	2%	.36 .15	0%	.12 .10
WIDL-CH-B^{1}	0%	.38 .19	12%	.32 .20	13%	.34 .13	33%	-.04 .06
Lapata, 2003	86%	.11 .03	67%	.12 .05	85%	.18 .00	24%	-.05 .06

Table 7.2: Evaluation of search algorithms for discourse coherence, for both EARTHQUAKES and ACCIDENTS genres, across the IBM_1^D, IBM_1^I, CM, and EB models. Performance is measured in terms of percentage of Estimated Search Errors (ESE), as well as quality of found realizations (average TAU and BLEU).

collection (henceforth called EARTHQUAKES) consists of Associated Press articles from the North American News Corpus on the topic of natural disasters. The second collection (henceforth called ACCIDENTS) consists of aviation accident reports from the National Transportation Safety Board's database.

For both collections, I used 100 documents for training and 100 documents for testing. A fraction of 40% of the training documents was temporarily removed and used as a development set, on which I performed the discriminative training procedure.

7.3.2 Evaluation of Search Algorithms

I evaluated the performance of several search algorithms across four stochastic models of discourse coherence: the IBM_1^D and IBM_1^I coherence models, the content model of Barzilay and

Lee (2004) (CM), and the entity-based model of Barzilay and Lapata (2005) (EB) (Section 6.2). I measure search performance using an Estimated Search Error (ESE) figure, which reports the percentage of times when the search algorithm proposes a sentence order which scores lower than the original sentence order (OSO). I also measure the quality of the proposed text realizations using TAU and BLEU, using as reference the original documents.

In Table 7.2, I report the performance of four search algorithms. The first three algorithms, WIDL-CH-A*, WIDL-CH-B^{100}, and WIDL-CH-B^1, are the WIDL-based search algorithms of Section 7.1, implementing A* search, histogram beam search with a beam of 100, and histogram beam search with a beam of 1, respectively. As a baseline, I compare the WIDL-CH algorithms with the greedy algorithm used by Lapata (2003). I note here that the comparison is rendered meaningful by the observation that Lapata's (2003) algorithm performs search identically with algorithm WIDL-CH-B^1 (histogram beam search with the beam set to 1), when setting the heuristic function for future costs to constant 0.

The results in Table 7.2 clearly show the superiority of the WIDL-CH-A* and WIDL-CH-B^{100} algorithms. Across all models considered, they consistently propose text realizations with scores at least as good as OSO (0% Estimated Search Error). As the original documents were coherent, it follows that the proposed text realizations also exhibit coherence. In contrast, the greedy algorithm of Lapata (2003) makes grave search errors. As the comparison between WIDL-CH-B^{100} and IDL-CH-HB^1 shows, the superiority of the WIDL-CH algorithms depends more on the admissible heuristic function h than in the ability to maintain multiple hypotheses while searching.

Model	TAU	BLEU	TAU	BLEU
	EARTHQUAKES		ACCIDENTS	
IBM_1^D	.38	.12	.41	.20
IBM_1^I	.32	.13	.40	.21
CM	.39	.12	.36	.15
EB	.19	.06	.12	.10
$\text{Log-linear}_{\text{uniform}}$	.34	.14	.48	.23
$\text{Log-linear}_{\text{maxTAU}}$	**.47**	.15	**.50**	.23
$\text{Log-linear}_{\text{maxBLEU}}$	.46	**.16**	.49	**.24**

Table 7.3: Evaluation of stochastic models for discourse coherence, for both EARTHQUAKES and ACCIDENTS genre, using WIDL-CH-B^{100}.

7.3.3 Evaluation of Log-linear Models

For this round of experiments, I held constant the search procedure (WIDL-CH-B^{100}), and varied the λ_m parameters of Equation 3.2. The utility-trained log-linear models are compared here against a baseline log-linear model $\text{log-linear}_{\text{uniform}}$, for which all λ_m parameters are set to 1, and also against the individual models. The results are presented in Table 7.3.

If not properly weighted, the log-linear combination may yield poorer results than those of individual models (average TAU of .34 for $\text{log-linear}_{\text{uniform}}$, versus .38 for IBM_1^D and .39 for CM, on the EARTHQUAKES domain). The highest TAU accuracy is obtained when using TAU to perform utility-based training of the λ_m parameters (.47 for EARTHQUAKES, .50 for ACCIDENTS), while the highest BLEU accuracy is obtained when using BLEU to perform utility-based training of the λ_m parameters (.16 for EARTHQUAKES, .24 for the ACCIDENTS). For both genres, the differences between the highest accuracy figures (in bold) and the accuracy of the individual models are statistically significant at 95% (using bootstrap resampling).

Overall performance	TAU	
	QUAKES	ACCIDENTS
Lapata (2003)	0.48	0.07
Barzilay & Lee (2004)	**0.81**	0.44
Barzilay & Lee (reproduced)	0.39	0.36
Barzilay & Lapata (2005)	0.19	0.12
Log-lin$_{\texttt{maxTAU}}$, WIDL-CH-B^{100}	0.47	**0.50**

Table 7.4: Comparison of overall performance (affected by both model & search procedure) of the WIDL framework with previous results.

7.3.4 Overall Performance Evaluation

The last comparison I provide is between the performance provided by the WIDL framework and previously-reported performance results (Table 7.4). I provide this comparison based on the TAU figures reported in (Barzilay & Lee 2004). The test data for both genres is held constant, therefore the figures can be directly compared. These figures account for combined model and search performance.

I first note that the reproduction of the model of Barzilay and Lee (2004) was not entirely successful. The CM model (Equation 6.4) has an average TAU figure of only .39 versus the original figure of .81 for EARTHQUAKES, and .36 versus .44 for ACCIDENTS. Given the estimated search errors figures for the decoding strategy, I attribute these differences to modeling issues. On the other hand, I reproduced successfully the model of Barzilay and Lapata (2005), and the average TAU figure is .19 for EARTHQUAKES, and .12 for ACCIDENTS[2]. The large difference on the EARTHQUAKES corpus between the performance of Barzilay and Lee (2004) and

[2]Note that these figures cannot be compared directly with the figures reported in (Barzilay & Lapata 2005), as they use a different type of evaluation. The EB model used here achieves the same performance as the original model in their evaluation setting.

the CM model is responsible for the overall lower performance (0.47) of the log-linear$_{\texttt{maxTAU}}$ model and WIDL-CH-B^{100} search algorithm, which is nevertheless higher than that of its component model CM (0.39). On the other hand, I achieve the highest accuracy figure (0.50) on the ACCIDENTS corpus, outperforming the previous-highest figure (0.44) of Barzilay and Lee (2004). These result empirically show that utility-trained log-linear models of discourse coherence outperform each of the individual coherence models considered.

7.4 Conclusions

In this chapter, I presented a WIDL-based framework that is capable of integrating various stochastic models of discourse coherence into a more powerful model that combines the strengths of the individual models. Important ingredients of this framework are the search algorithms based on WIDL-expressions, which provide a flexible way of solving discourse generation problems using stochastic models. The WIDL-CH generation algorithms are fundamentally different from previously-proposed algorithms for discourse generation (Mellish *et al.* 1998; Karamanis & Manurung 2002; Lapata 2003; Althaus, Karamanis, & Koller 2005).

For each of the coherence model combinations that I have utility trained, I obtained improved results on the discourse ordering problem compared to the individual models. This is important for two reasons. These improvements can have an immediate impact on multi-document summarization applications (Barzilay, Elhadad, & McKeown 2002). Also, the WIDL framework provides a solid foundation for subsequent research on discourse coherence models and related applications.

Chapter 8

Intersecting WIDL-expressions with n-Gram and Syntax-based Language Models

WIDL-expressions are a convenient formalism for compactly representing very large sets of possible string realizations, weighted according to input biases expressed using the distribution functions defined for the $\vee$ and $||$ operators. WIDL-expressions do not usually integrate target-language biases, as WIDL-expressions are agnostic to any type of language theory. To capture language biases in the generation process, I use a log-linear combination of WIDL-expression probability distributions with n-gram language probability distributions, for which I define algorithms capable of finding the best realization (Chapter 3).

However, n-gram language models are not the only option to express target-language biases. There have been numerous studies in the literature concerning the usefulness of language models based on hierarchical, syntax-oriented structures, for various tasks such as speech recognition (Chelba & Jelinek 1998; Charniak 2000), natural language generation (Bangalore & Rambow 2000a; Daume *et al.* 2002), and machine translation (Charniak, Knight, & Yamada 2003;

Habash 2004; Och *et al.* 2004). The results, however, are inconclusive. A structural language model based on syntactic prefixes was shown to yield improvements in speech recognition accuracy over a trigram language model baseline (Chelba 2000), and a language model based on a generative model of syntax was shown to be helpful for generation (Daume *et al.* 2002), as well as in improving the performance of a syntax-oriented machine translation system (Charniak, Knight, & Yamada 2003). On the other hand, a structural n-gram language model based on lexical dependecies was found to yield no improvements in the accuracy of a generation-heavy hydrid machine translation system (Habash 2004), while a language model based on a generative model of syntax was not helpful in improving the performance of a state-of-the-art, phrase-based machine translation system (Och *et al.* 2004).

In this chapter, I will focus on the integration of a syntax-based language model, based on the generative model of syntax described by Collins (2003), with WIDL-based probability distributions and n-gram language model probability distributions. The evaluation results show that, for short sentences, the integration of syntax-based and n-gram language models yields superior results on a word reordering task compared to the performance of an n-gram language model alone. On the other hand, the addition of the syntax-based language model does not result in superior performance for longer sentences, on the same task.

8.1 A Language Model based on a Generative Model of Syntax

Certain syntax-based language models use the notion of a phrase-structure syntax tree associated with the sequence $s = w_1 \ldots w_m$ to arrive at a language model probability $P(s)$. A

phrase-structure syntax tree can be produced by a trained linguist annotator according to some specific annotation guidelines (Marcus, Santorini, & Marcinkiewicz 1993). It can also be produced automatically by a parser. The most accurate parsers to date are statistical, lexicalized parsers that use a generative model to find the most probable syntax tree t associated with a sequence s (Collins 1999; Charniak 2000). Instead of finding the tree t which has a maximum $P(t|s)$ value (the best tree t given sentence s), these parsers find the tree t which has a maximum $P(t, s)$ value, which is equivalent with the former because of Equation 8.1:

$$\arg\max_t P(t|s) = \arg\max_t P(t, s) \tag{8.1}$$

From a language modeling perspective, the advantage of having a generative model which assigns $P(t, s)$ values to every (t, s) pair is that one can define a language model probability $P(s)$ for the sequence s by summing over all the possible trees t, as in Equation 8.2:

$$P(s) = \Sigma_t P(t, s) \tag{8.2}$$

where $P(t, s) = 0$ if $\mathsf{yield}(t) \neq s$. Throughout this chapter, I will use the term syntax-based language models to refer to language model of the type defined by Equation 8.2.

Syntax-based language models hold the promise of assigning better $P(s)$ values than n-gram language models, because they characterize sequence s globally (via the syntax trees

over which they sum up), whereas n-gram language models characterize sequence s only locally, using a window of size n. Combining together an n-gram language model and a syntax-based language model holds the promise of exploiting both the advantages of n-gram language models (well-studied smoothing techniques (Goodman 2001), scalability to large amounts of training data, huge amounts of training instances, publicly-available toolkits (Stolcke 2002)) and the advantages of syntax-based language models (well-studied models of language usage (Haegeman 1994), available generative models of syntax (Charniak 2001; Collins 2003; Bikel 2004)).

Interpolating WIDL Probability Distributions with a Syntax-based Language Model. Integrating a syntax-based language model into a log-linear combination of WIDL probability distributions and n-gram probability distributions is achieved using an additional feature function h_{SB} in Equation 3.1 (Chapter 3). The problem of finding the most probable realization e under this extended model can be formulated as shown in Equation 8.3:

$$\arg\max_{e} P(e) = \arg\max_{e}(\Sigma_{m=0}^{M}\lambda_m \log h_m(e) + \lambda_{SB} \log h_{SB}(e)) \tag{8.3}$$

However, formulating algorithms that can solve Equation 8.3 is a much more difficult proposition. Because syntax-based language models characterize strings using (hypothesized) global hierarchical structures, partial estimates of their scores are difficult to obtain and integrate with local estimates such as those given by n-gram language models. The solution usually adopted in

this case is to get an approximation of the search space considered by Equation 8.3, by considering the k best-ranking hypotheses when ignoring the $h_{SB}(e)$ term. This k-best approximation is consequently scored and ranked according to the full Equation 8.3 (Charniak, Knight, & Yamada 2003; Och *et al.* 2004). The disadvantage of this solution is that it greedily restricts the search space to k hypotheses that score well under a weaker model, and therefore it loses hypotheses that could have scored better under the full model.

In this chapter, I propose a new solution to the problem of finding the best realization under models for which integrated search cannot be achieved. Although I present the solution in the context of integrating n-gram language models with syntax-based language models, the proposed algorithm (Section 8.2) is general enough to be applicable to other similar scenarios. The main advantage of this algorithm is that it does not greedily restrict the search space. Instead, it approximates integrated search using a lazy integration of model scores.

8.2 An Algorithm for Intersecting WIDL-expressions with n-Gram and Syntax-based Language Models

Algorithm WIDL-NG_SBLM-A* This algorithm uses the same idea of incremental unfolding of a WIDL-graph G_ω and A* search strategy as WIDL-NGLM-A* (Chapter 3, Section 3.4), and is described in Figure 8.1. If n is the maximum size of the context required by any of the language models in $\overline{NG}$, the algorithm computes incrementally the relation UNFOLD$^n_\omega$ for G_ω (see Chapter 2) at line 4, using a current $active$ state. The set of newly UNFOLDed states is called $unfold$. Using Equation 8.3, the algorithm EVALUATEs the current $P(e)$ costs for the

unfold states, using only the $\overline{NG}$ models and $\overline{\lambda}$. I call these costs $g_{\overline{NG}}$. EVALUATE also uses an admissible heuristic function (Russell & Norvig 1995) to compute future (admissible) costs for the *unfold* states. The future costs for the $\overline{NG}$ models are computed using Equation 3.4 (Chapter 3) – I call these costs $h_{\overline{NG}}$; the costs for the SB model are computed using admissible costs, computed as described in the next section – I call them h_{SB}; the total cost of a state in *unfold* is computed as $f = g_{\overline{NG}} + h_{\overline{NG}} + h_{SB}$. One should note the absence of a g_{SB} term (current cost under the SB model), as the algorithm does not have access to partial syntax-based scores in the absence of a (partial) tree; to compensate, the term h_{SB} provides total admissible costs under model SB (Section 8.3).

To control the incremental unfolding, the algorithm uses a priority queue, Q, in which the states from *unfold* are PUSHed (line 8), sorted according to the total cost f associated with each state by the EVALUATE function. Each PUSHed state is accompanied by a *synflag* value, which indicates whether the state is FINAL (i.e., final state in the pFSA A^n_ω corresponding to G_ω) or not.

The new active state is the state POPed out from the top of Q (the lowest cost state), together with its *synflag* value. If *synflag* is 0 (*active* state is non-final), the next iteration starts. If *synflag* is 1 (*active* state is final), the SYNEVALUATE function is called (line 11). This function extracts the path from the *active* state to the initial state, and computes a g_{SB} cost (including λ_{SB}) based on syntax trees proposed by a parser (which runs on the full path). As *active* is a final state, the future costs are 0, and therefore the total cost is $f = g_{\overline{NG}} + g_{SB}$, that is, the actual cost of state *active* under the considered $\overline{NG}$ and SB combination (Equation 8.3). This

WIDL-NG_SBLM-A$^*(G_\omega, \overline{NG}, \overline{\lambda}, SB, \lambda_{SB})$

```
 1  active ← [vs^{G_ω}]
 2  flag ← 1 Q ← ∅
 3  while flag
 4        do unfold ← UNFOLD(active, G_ω, NG)
 5           EVALUATE(unfold, NG, λ)
 6           for each state in unfold
 7                do if FINAL(unfold, G_ω)
                        then synflag ← 1
                        else  synflag ← 0
 8                   PUSH(Q, ⟨state, synflag⟩)
 9           ⟨active, synflag⟩ ← POP(Q)
10           if synflag = 2
               then flag ← 0
               else  if synflag = 1
11                       then SYNEVALUATE(active, SB, λ_SB)
12                            PUSH(Q, ⟨active, 2⟩)
13  return active
```

Figure 8.1: Pseudo-code for intersecting a WIDL-graph G_ω with n-gram language models $\overline{NG}$ and a syntax-based language model SB, using incremental unfolding and A* search.

state is again PUSHed in Q (line 12), with *synflag* set to 2. Finally, if the current POPed *synflag* value is 2, the algorithm finishes by returning the current *active* state.

Similar with the WIDL-NGLM-A* algorithm, I also use a k-steps version (see Chapter 3, Section 3.4) of the WIDL-NG_SBLM-A* algorithm. The k-steps version is a faster, greedy version, which finds good, approximative solutions in the case when computing the optimal solution requires too many resources (space and time–wise).

8.3 p-Admissible Heuristics for Syntax-based Language Models

A possible way to compute admissible estimates for a sequence $s = w_1 \dots w_m$, under a probabilistic model PM over some domain D, is to abstract s in some way into a $\alpha(s)$ summary, and propose as admissible cost the maximum model probability for all s' sequences that fit the summary:

$$h_{PM}(s) = \max_{s' \in D, \alpha(s)=\alpha(s')} g_{PM}(s') \tag{8.4}$$

where $g_{PM}(s')$ is the actual probability of sequence s' under model PM. This equation is similar with the one used by Klein and Manning (2003) to define admissible heuristics for parse trees.

The summary function α is said to partition (factorize) the domain D according to the equivalence classes $\alpha(s)$, and the resulting set is denoted D/α. If the summary gives no information about s, then $|D/\alpha| = 1$ (all strings are in the same equivalence class), and therefore the estimate is a constant estimate which does not offer any computational advantage. If the summary preserves all the information about s, then $|D/\alpha| = |D|$ (every string has its own equivalence class), and therefore $h_{PM}(s) = g_{PM}(s)$, the ideal estimate.

In practice, however, the summary $\alpha(s)$ should provide some information about s such that it is reasonable to pre-compute all $|D/\alpha|$ estimates that are of interest. For instance, if the summary function α is defined as $\alpha(w_1 \dots w_m) = \mathsf{length}(w_1 \dots w_m) = m$ (the number of words in the string), and one knows a-priori that only strings of length 1 to N are of interest,

then the problem of computing admissible estimates reduces to estimate one value for each element in D/length, that is, estimate N values, one for each possible length.

For probabilistic models PM which cannot easily decompose their $g_{PM}(s)$ value, and strings s over large vocabularies, computing the true $\max$ value for Equation 8.4 might not be feasible. A lexicalized syntax-based language model SB , which assigns $g_{SB}(s)$ values according to Equation 8.2 to strings over a large vocabulary set, is such an example. For such cases, I introduce the notion of p-admissible cost for s, where p is the probability that the cost function h really is admissible for string s under model PM, that is, $h_{PM}(s) \geq g_{PM}(s)$.

Definition 11 *Consider a finite alphabet Σ, a probability distribution δ over a domain $D \subseteq \Sigma^*$, and a summarization function α defined over D. For any random $s \in D$ and any random finite set $Tr \subseteq D$, the value*

$$h_\delta(s|\,Tr) = \max_{s' \in Tr, \alpha(s)=\alpha(s')} \delta(s')$$

is called a p-admissible cost for s given Tr, where $p = \frac{|Tr|}{|Tr|+1}$.

The set Tr is called the training corpus. The above definition extends trivially to a set Te (called test corpus). I write $h_\delta(Te|\,Tr)$ for the set $\{h_\delta(s|\,Tr) \mid s \in Te\}$, and call it the set of p-admissible costs for Te given Tr, where p is computed as $p = \frac{|Tr|}{|Tr|+|Te|}$. One should note that the size of the set of admissible costs, $|h_\delta(Te|\,Tr)|$, is independent of the sizes of Te and Tr, as it is always $|D/\alpha|$.

Definition 11 provides an empirical strategy for computing heuristics that can be more useful than using a constant 0 cost function as an admissible heuristics, and more efficient to compute than using the ideal estimate as the admissible heuristic cost. It does so by providing two parameters, the summary function α and the training set Tr, which can be set such that one achieves a balance between the cost of computing the heuristics and their usefulness (given by the choice of α) on one hand, and the theoretical guarantees associated with their admissiblity (given by the choice of Tr), on the other hand.

As an example of the practical applications of Definition 11, consider a syntax-based language model that uses Collins's (2003) generative Model 2 for computing $P(t, s)$, and computes the marginal over all trees by considering only the top scoring tree (see Section 8.1). The parameters of this model are trained using the syntax trees from the Penn Treebank, a corpus of about 40,000 sentences from the Wall Street Journal, annotated with syntax trees by human annotators (Marcus, Santorini, & Marcinkiewicz 1993). I call this model SB. Also, from a Wall Street Journal collection which does not include the sentences used in the Penn Treebank, I extract a random sample of 1,000 sentences as set Te (denoted Te-1k), and a random sample of 40,000 sentence as set Tr (denoted Tr-40k). In Figure 8.2, I plot the minimum $\mathrm{SB}(s)$ costs ($-\log$ probabilities) for the Te set (the Te-1k curve) and the Tr set (the Tr-40k curve), using as summary function $\alpha = \mathsf{length}$ (string length), for lengths between 1 and 40[1].

According to the definition, $h_{\mathrm{SB}}(\text{Te-1k}|\text{Tr-40k})$ are p-admissible costs for Te-1k given Tr-40k, with $p = \frac{40000}{41000} = 0.975$. This means there is a 97.5% probability that, for some random

[1]The values in the plot are smoothed such that the curves are monotonically increasing.

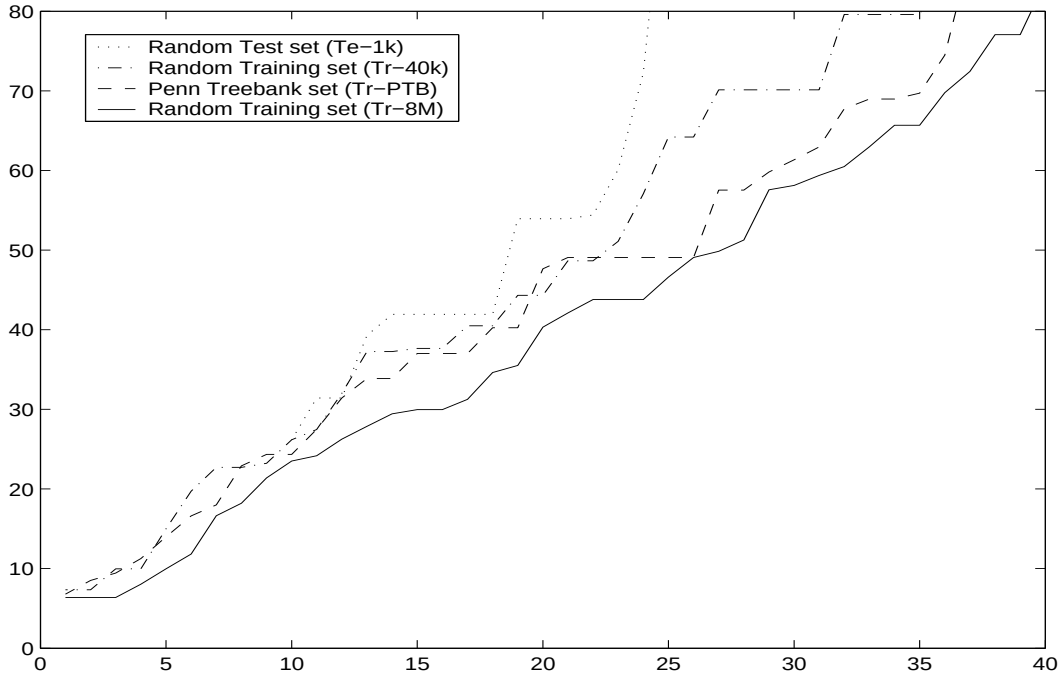

Figure 8.2: Minimum $\mathrm{SB}(s)$ costs ($-\log$ probabilities) for a 1000 sentence corpus (the Te-1k curve), a 40,000 sentence corpus (the Tr-40k curve), the Penn Treebank corpus (the Tr-PTB curve), and a 8 million sentence corpus (the Tr-8M curve).

$s \in$Te-1k, the value $h_{\mathrm{SB}}(s|\text{Tr-40k})$ is admissible, and a 2.5% probability that it is not. The summary function length partitions the domain D into 40 equivalence classes, and therefore the size of the set $h_{\mathrm{SB}}(\text{Te-1k}|\text{Tr-40k})$ is 40. It follows that there exists $\frac{2.5}{100} \times 40 = 1$ value for which the p-admissible estimate is not admissible. Indeed, as one can see using the graphs in Figure 8.2, the minimum values for Te-1k are all higher than the minimum values for Tr-40k, except for the class of strings of length 12. For some $s \in$Te-1k having length 12, the cost is smaller than the cost of any $s' \in$Tr-40k of length 12, which implies that the estimate $h_{\mathrm{SB}}(s|\text{Tr-40k})$ is not admissible. Table 8.1 shows this minimum cost string $s \in$Te-1k having cost 31.41

Corpus	Counts for $\alpha(s) = 12$	Best string for $\alpha(s) = 12$	Cost $(-\log p)$
Te-1k	47	revenue rose 8 % to $ 467.7 million from $ 433.5 million	31.41
Tr-40k	1143	revenue rose 11 % to $ 99.3 million from $ 89.4 million	32.00
Tr-PTB	1233	revenue rose 15 % to $ 534.3 million from $ 464.7 million	31.43
Tr-8M	214873	in the year - earlier period , the company earned $ 1.33	26.26

Table 8.1: Minimum $\mathrm{SB}(s)$ costs ($-\log$ probabilities) for sentences of length 12 from a 1000 sentence corpus (Te-1k), a 40,000 sentence corpus (Tr-40k), the Penn Treebank corpus (Tr-PTB) and a 8 million sentence corpus (Tr-8M).

(best out of 47 instances), as well as the best cost string of length 12 from set Tr-40k, having cost 32.00 (out of 1143 instances).

From Definition 11, it follows that p-admissible values can be made increasingly closer to true admissible values in two ways. First, by using as training set not a random sample, but a carefully chosen set which maximizes the probability of each string in this set. In our case, one possible choice is the set containing the strings from the Penn Treebank, on which the SB model is trained. As the plot in Figure 8.2 shows (the Tr-PTB curve), the admissible costs computed in this fashion tend to be smaller, although they are not guaranteed to be admissible (see third entry in Table 8.1 for a p-admissible estimate which is not admissible for the set Te-1k).

Second, one can increase the size of the training set Tr up to a value which gives p-admissible costs with probability p of reasonable confidence. For instance, a training set Tr-8M of size 8 million will give, for a test instance Te of size 1000, a probability $p = \frac{8{,}000{,}000}{8{,}001{,}000} = 99.9875\%$ of admissibility for its p-admissible values. That is, there is a probability of 0.0125% that a value $h_{\mathrm{SB}}(s|\text{Tr-8M})$, $s \in Te$, is not admissible. Given that $|D/\mathsf{length}|$ is 40, it follows that one can expect $\frac{0.0125}{100} \times 40 = 0.005$ actual values for which the p-admissible estimate is

not admissible. This is a comfortable enough value so that one can assume that, in fact, all the p-admissible estimates are truly admissible. Figure 8.2 confirms this for the Te-1k corpus and the Tr-8M corpus considered, as the curve for the minimum Tr-8M values is comfortably lower than the curve for the minimum Te-1k values. Also, Table 8.1 (last row) confirms, for strings of length 12, the existence of an admissible cost of 26.26 for the Te-1k corpus.

p-Admissible Heuristics for WIDL-expressions Algorithm WIDL-NG_SBLM-A* computes $h_{\mathrm{SB}}(s)$ at each call of the EVALUATE function, for all strings s in $\mathsf{dom}(\sigma_{\mathrm{widl}}(\omega))$ of the input ω expression. For a summary function α that compute value $\alpha(s)$ at each EVALUATE step, one can use Equation 8.4 to compute an admissible cost for s.

For any WIDL-expression ω, an important piece of information readily available is a lower-bound on the size of all strings encoded by ω, computed as $\mathsf{shortest}(\omega)$ (Chapter 2, Definition 9). With the summary function α defined as $\mathsf{shortest}$ for all strings $s \in \mathsf{dom}(\sigma_{\mathrm{widl}}(\omega))$, Equation 8.4 defines an admissible cost $h_{\mathrm{SB}}(s)$. As computing the $\max$ function is not feasible, I approximate the cost $h_{\mathrm{SB}}(s)$ by a p-admissible cost $h_{\mathrm{SB}}(s|\,Tr)$, for some training set Tr which yields an acceptable p value.

8.4 Evaluation of Combinations between n-Gram and Syntax-based Language Models

Perplexity is an intrinsic measure on the performance of a language model, and it has been shown (Charniak 2001) that a linear combination of n-gram and syntax-based language models

has lower perplexity on a held-out test set than either of the individual models. Ultimately, however, a language model is better if it can be shown to improve performance using an extrinsic measure, related to a real task, such as speech recognition and machine translation. Unfortunately, integrating language models within speech recognition or machine translation systems is a laborious process. Using the WIDL framework described in this dissertation, I propose a direct way to extrinsically measure the performance of language models, using a word reordering task.

8.4.1 Evaluation Setting

In the word reordering task, a pre-selected bag of words needs to be arranged in a sequence which maximizes some specific quality. I define this quality to be the correctness of the sequence from a syntactic, semantic, and even pragmatic point of view; in short, what a native speaker would regard as being sensible language usage. To ensure that at least one correct order exists for some pre-selected bag of words, I take it to be the bag of words of some existing sentence. In this way, one correct order of the words is guaranteed to exist (the original sentence order, or OSO).

I measure the accuracy of a language model by comparing the probability assigned by the model to various sequences against the probability assigned by the model to OSO. If the language model assigns the highest probability to OSO, one can be confident that the language model does the right thing. If the language model assigns the highest probability to some other order, especially when this order is completely different from that of OSO, it is likely that it

is due to a modeling error, as it is on average unlikely (although sometimes possible) that a different but correct sentence using the same words exists.

As a bag of words of size n has $n!$ possible sequences, simply enumerating all the sequences and then scoring them is impractical, if not impossible (for a bag of words of size 22, the average sentence length for Wall Street Journal text, there are approximately 10^{21} possible sequences). Therefore, a solution to the word reordering task also needs efficient algorithms that are capable of finding the highest probability sequence (or a good approximation), without trying to evaluate too many of them (since trying too many means taking too much time). As a consequence, I also need to estimate the errors introduced by the search procedure. In the case when the algorithm outputs a sequence for which the language model assigns a lower probability than the probability of OSO, one has a search error (since finding the original sentence order would have resulted in a higher probability sequence). If the algorithm outputs a sequence for which the language model assigns a probability equal or higher than that of OSO, then I consider the algorithm to have searched successfully.

8.4.2 Evaluation of Language Model Combinations

I use the algorithms described in Section 8.2 to measure whether syntax-based language models provide increased performance on the word reordering task over simply using n-gram language models. I run two experiments. The first experiment uses bag-of-words of small sizes, between three and seven words (excluding the final punctuation). The second experiment uses bag-of-words of larger sizes, between 10 and 25 words (excluding the final punctuation). Both

experiments use a blind test set of 1000 sentences, and a development/tuning set of another 1000 sentences.

For both experiments, I use as a baseline a trigram language model trained on about 8 million sentences (185 million words) from the Wall Street Journal, smoothed using Kneser-Ney smoothing. The syntax-based language model I employ uses Collins's (2003) generative Model 2 for computing $P(t, s)$, and computes the marginal over all trees by considering only the top scoring tree (see Section 8.1). One instance of this syntax-based language model, called SB, is trained using the trees from the Penn Treebank (Marcus, Santorini, & Marcinkiewicz 1993). A second instance of the syntax-based language model, called SB$^+$, bootstraps its training data by using, in addition to the Penn Treebank, a corpus of about 160,000 sentences from the Wall Street Journal, for which syntax trees were produced using the same generative model $P(t, s)$ and my implementation of Collins's parsing procedure (Collins 2003; Bikel 2004).

Each instance of the syntax-based language model is log-linearly combined with the trigram language model, using two linear coefficients that are discriminatively trained (Och 2003) to maximize the performance of the log-linear combination using BLEU (Papineni *et al.* 2002) as utility function. BLEU is also one of the metrics used to evaluate the accuracy on the task against the original sentence order, and ranges from 0 (the worst) to 1 (the best). I report here BLEU% scores (BLEU multiplied by 100), which range from 0 to 100. The other metric is identity percentage (ID), the percentage of sentences that perfectly reconstruct the original sentence order.

Model Combination		eSErr	ModErr	ID	BLEU%	95%-conf.
3G=1	SB=0	0%	32.5%	67.1%	80.83	78.99 - 82.75
3G=0.56	SB=0.44	1.6%	26.9%	71.5%	82.36	80.44 - 84.22
3G=0.73	SB^{+}=0.27	1.1%	26.4%	72.9%	83.68	81.85 - 85.45

Table 8.2: Evaluation of trigram and syntax-based language model combinations using bags-of-words of size 3-7. Metric eESerr measures the percentage of realizations that score lower than Original Sentence Order (OSO), metric ModErr measures the percentage of realizations that score higher than OSO, metric ID measures the percentage of realizations identical to OSO, and BLEU% provides BLEU scores against OSO.

For the experiment involving bag-of-words of size 3-7, I use the WIDL-NG_SBLM-A* algorithm. The results are presented in Table 8.2. The baseline model has an ID score of 67.1% and a BLEU% score of 80.8. There are 0 search errors introduced by the search algorithm, and 32.5% modeling errors.

The first trigram and SBLM combination, using human-produced syntax trees, was discriminatively trained on the tuning set (bag-of-words of size 3-7) using BLEU, and the linear coefficients were 0.56 for the trigram and 0.44 for the SBLM. The ID score of this combination is 71.5%, and the BLEU% score is 82.36. The difference between this BLEU score and the baseline BLEU score is not statistically significant. There are 1.6% estimated search errors introduced by the search algorithm, and 26.9% modeling errors.

The second trigram and SBLM combination, using both human-produced syntax trees and automatically created syntax trees, was discriminatively trained on the tuning set (bag-of-words of size 3-7) using BLEU, and the linear coefficients were 0.73 for the trigram and 0.27 for the SBLM. The ID score of this combination is 72.9%, and the BLEU% score is 83.68. The difference between this BLEU score and the baseline BLEU score is statistically significant

	Order	-log p		
		3G	SB	LogLin
OSO	’ oh , please , hubie	22.3	79.2	47.3
HYP	oh , hubie ’ please ,	23.0	43.5	32.0
OSO	better bart than madonna , i say	28.5	58.3	41.6
HYP	i say better than bart madonna ,	25.9	40.5	32.3
OSO	and is this a good thing	17.5	32.9	24.3
HYP	and this is a good thing	13.4	25.4	18.7

Table 8.3: Examples of original sentence order (OSO) and best realization found (HYP) and their -log probability (lower is better) under trigram (3G) and syntax-based (SB) language model log-linear (LogLin) combination, for bag-of-words of size 3-7.

at 95% using bootstrap resampling. There are 1.1% estimated search errors introduced by the search algorithm, and 26.4% modeling errors.

To get a sense of the type of modeling errors that occurred, I present in Table 8.3 three test instances for which the difference between the OSO probability and the probability of the best realization (HYP) found is large. As one can observe, the different realizations are not necessarily worse than OSO, and some are correct but represent biases of the language models due to the training data (such as the preference for assertive versus interrogative formulations).

In conclusion, the improvements in ID and BLEU scores, as well as the drop in the percentage of modeling errors, lead us to conclude that, for small bag-of-words, a well-tuned combination between a trigram language model and a (bootstrapped) syntax-based language model provides increased language model performance.

For the experiment involving bag-of-words of size 10-25, I use the WIDL-NG_SBLM-A$^*_3$ algorithm (the k-steps version of the A* algorithm, with $k = 3$). The results are presented in

Model Combination		eSErr	ModErr	ID	BLEU%	95%-conf.
3G=1	SB=0	6.3%	84.0%	9.5%	57.30	55.88 - 58.61
3G=0.94	SB=0.06	8.0%	82.1%	10.1%	57.66	56.25 - 58.99
3G=0.95	SB^{+}=0.05	5.9%	81.9%	10.2%	58.07	56.67 - 59.42

Table 8.4: Evaluation of trigram and syntax-based language model combinations using bags-of-words of size 10-25. Metric eESerr measures the percentage of realizations that score lower than Original Sentence Order (OSO), metric ModErr measures the percentage of realizations that score higher than OSO, metric ID measures the percentage of realizations identical to OSO, and BLEU% provides BLEU scores against OSO.

Table 8.4. The baseline has an ID score of 9.5% and a BLEU% score of 57.3. There are 6.3% search errors introduced by the search algorithm, and 84.0% modeling errors.

The first trigram and SBLM combination, using human-produced syntax trees, was discriminatively trained on the tuning set (bag-of-words of size 10-25) using BLEU, and the linear coefficients were 0.94 for the trigram and 0.06 for the SBLM. The ID score of this combination is 10.1%, and the BLEU% score is 57.66. The difference between this BLEU score and the baseline BLEU score is not statistically significant, at 95% confidence. There are 8.0% estimated search errors introduced by the search algorithm, and 82.1% modeling errors.

The second trigram and SBLM combination, using both human-produced syntax trees and automatically created syntax trees, was discriminatively trained on the tuning set (bag-of-words of size 10-25) using BLEU, and the linear coefficients were 0.95 for the trigram and 0.05 for the SBLM. The ID score of this combination is 10.2%, and the BLEU% score is 58.07. The difference between this BLEU score and the baseline BLEU score is not statistically significant, at 95% confidence. There are 5.9% estimated search errors introduced by the search algorithm, and 81.9% modeling errors.

	Order	-log p		
		3G	SB	LogLin
OSO	a show hostess rolls a huge foam ball down an incline at 10 college students dressed as bowling pins , knocking nine to the snow	88.9	173.0	94.3
HYP	college students show dressed as a hostess knocking down the snow ball rolls at a huge bowling pins , an incline foam nine to 10	74.1	168.2	80.1
OSO	could two things so apparently dissimilar as a thought and neural fi ring really be identical	61.7	101.7	64.3
HYP	and so could really be a neural apparently thought things as dissimilar fi ring two identical	47.9	105.8	51.6
OSO	the sleek lhs and the new new yorker are a marked departure from some of chrysler 's past offerings , such as the imperial	59.8	125.3	64.0
HYP	some of the new offerings , such as a marked departure from the past are the chrysler imperial 's sleek new yorker and lhs	47.4	113.4	51.7

Table 8.5: Examples of original sentence order (OSO) and best realization found (HYP) and their -log probability (lower is better) under trigram (3G) and syntax-based (SB) language model log-linear (LogLin) combination, for bag-of-words of size 10-25.

Again, to get a sense of the type of modeling errors that occurred, I present in Table 8.5 three test instances for which the difference between the OSO probability and the probability of the best realization (HYP) found is large. In contrast with the modeling errors for small bag-of-words, one can observe that the SBLM fails to impose global correctness, and often favors incoherent orders for which better scoring syntax trees are created. This tendency is also captured by the low interpolation coefficient assigned by the discriminative training procedure. Even if the SBLM sometimes prefers the OSO order (second example in Table 8.5), the low interpolation coefficient attenuates the preference.

The minor improvements in ID and BLEU scores, as well as the high percentage of modeling errors, even for well-tuned combinations between a trigram language model and a (bootstrapped) syntax-based language model, leads us to conclude that, for large bag-of-words,

adding a syntax-based language model based on a generative parsing model does not increase language model performance significantly.

8.5 Conclusions

In this chapter, I have used the WIDL generation framework to measure the contribution of a hierarchic, syntax-based language model to a generation task, compared to the baseline performance of a flat, n-gram–based language model. This contribution can be measured directly by changing the stochastic language model used by the generation engine. However, finding the best realization under a stochastic language model that uses hidden hierarchical structures is not straightforward. Toward this end, I have defined a new notion of forward estimation, called p-admissibility, and a new A*-based algorithm based on lazy integration of model scores.

The results of the evaluations performed indicate that a language model based on a generative model of syntax improves generation accuracy over a baseline trigram language model when the generated sentences are short, but does not currently offer significant improvements for generating long sentences.

Chapter 9

Previous Work in Natural Language Generation

Traditionally, natural language generation has been approached as a concept-to-text realization task. The concept-to-text approach assumes that a set of “concepts”, or “meanings”, are explicitly available for generation, and concentrates on how to arrive at correct language usage that communicates the desired meaning. Computationally, this approach has often been divided into three stages: content selection, sentence planning, and sentence realization (Reiter 1994). Content selection, or content planning, is the task of deciding what the communication act is about, or simply put, “what to say”. A part of the content planning stage is content ordering, which decides, in tandem with content selection, the order in which the various pieces of information are communicated, often following constraints imposed by language theories, such as Rhetorical Structure Theory (Mann & Thompson 1988). Once the content has been determined, sentence planning is the task of deciding “how to say it”. It makes use of various pieces of information about the communication context and order, and decides whether to use full names or pronouns when referring to various entities, whether some piece of information is redundant and can be elided, etc. The last task is sentence realization, during which one needs to perform, among

other sub-tasks, lexical choice for content words, insert closed-class function words, provide defaults for under-specified syntactic features, apply agreement and grammatically constraints, determine the linear precedence in the final realization, etc.

More recently, natural language generation has also been approached as a text-to-text realization task. The text-to-text approach is more focused on specific applications that can benefit from natural language generation techniques, such as multi-document summarization (Barzilay 2003). This approach makes no assumptions about the concepts or meanings to be communicated, which are opportunistically and only implicitly extracted from textual inputs. Computationally, this approach differs from the concept-to-text approach, in that the content selection and content ordering stages are torn apart, whereas sentence planning and sentence realizations are grouped together. In summarization, for instance, content selection is driven by salience criteria, and content ordering is afterwards computed depending on cohesion, temporal dependencies, etc., that exist between the information units selected (Barzilay, Elhadad, & McKeown 2002). Also, text-to-text sentence realizations techniques, such as the noisy channel model (Brown *et al.* 1993) and information fusion (Barzilay 2003) perform both higher-level decisions about sentence-level information content (entity reference, inclusion/exclusion of material, etc.) and lower-level decisions (lexical choice between synonyms, usage of closed-class words, etc.) simultaneously, based on various criteria that the output realization must satisfy.

There are, however, many similarities between the concept-to-text approaches and text-to-text approaches to sentence realization, as discussed in Section 9.1. Also, the main characteristics of content ordering techniques used in concept-to-text and text-to-text approaches are discussed in Section 9.2.

9.1 A Computational Perspective on Sentence Realization

9.1.1 Computational Requirements for Sentence Realization

Most of the NLG systems recently proposed in the literature take a hybrid approach to generation, by combining human-crafted symbolic knowledge with automatically acquired statistical knowledge (Knight & Hatzivassiloglou 1995; Bangalore & Rambow 2000b; Langkilde-Geary 2002). These systems are more robust, achieve higher accuracy, and provide wider coverage than the older, purely symbolic approaches to generation (Meteer *et al.* 1987; Matthiessen & Bateman 1991).

A typical hybrid system uses two processing stages. In the first stage, the input meaning, which is usually some logical/semantic representation, is transformed into a representation that encodes a large set of candidate sentences/realizations. This transformation is typically carried out by a symbolic, hand-written grammar, which has the advantage that it is small and easy to develop and maintain. The disadvantage of using such a grammar is that this first stage badly overgenerates. That is, many candidates that are ungrammatical or otherwise less desirable cannot be distinguished from more desirable ones.

In the second stage, the collection of candidates is evaluated by mechanisms that are more informed about the desired properties of the strings in the target natural language. The language knowledge used at this stage, such as n-gram language models, is usually automatically extracted from corpora, and therefore is considerably less expensive than hand-written, symbolic knowledge. Moreover, such language model sources come also equipped with the ability to assign probabilities to language events.

Together, the two stages can properly operate in a hybrid system if certain properties hold:

A. The meaning representation formalism allows for a compact representation of a large set of candidate realizations; and

B. The generation mechanism employs efficient algorithms for intersecting, scoring, and ranking the candidate set with rich language knowledge resources.

Requirement **A** allows one to deal with underspecification and ambiguity, two omnipresent characteristics of the natural language generation task. Requirement **B** allows systems to graciously handle increasingly difficult input, and also exploit richer models of language well-formedness, which allows linguistic knowledge about how natural language realizations are formed to be computationally exploited. In what follows, I will discuss some of the generic NLG systems proposed to date, and also some application-specific proposals, from the perspective of the requirements **A** and **B** presented above.

```
( R /   release
                : tense past
                : voice passive
        AGENT   ( P / police )
        PATIENT ( C / prisoner
                : number plural ))
```

Figure 9.1: Possible HALogen input for the sentence the prisoners were released by the police.

9.1.2 Sentence Realization in Statistical Natural Language Generation

Nitrogen and HALogen. The first statistical NLG system proposed in the literature was Nitrogen (Knight & Hatzivassiloglou 1995). HALogen (Langkilde-Geary 2002) later modified and extended Nitrogen. As most of the NLG systems following on its steps, Nitrogen was a hybrid system, combining human-crafted symbolic knowledge with automatically acquired statistical knowledge. Experiments done with Nitrogen and HALogen showed that statistical knowledge derived from corpora, in the form of n-gram language models, greatly reduce the need for hand-crafted knowledge.

A typical input taken by a system like HALogen is shown in Figure 9.1. From a text-to-text generation perspective, this type of input is clearly difficult to obtain, as it requires specifying semantic/syntactic relations between the concepts/words involved. Such relations may be computed for concepts/words appearing inside one sentence, but are problematic to determine across multiple sentences, or across multiple documents.

I now turn the discussion to the representation formalisms used by these systems. In the case of Nitrogen, the representation formalism for the set of possible realizations is that of acyclic finite automata, also called word lattices. Word lattices have been introduced by the speech

recognition community as a natural way to encode the multiple word-choice corresponding to a speech wave sequence, and the temporal precedence encoded in the sound waves. The advantage of using word lattices is that there are algorithms to efficiently process them. For a lattice with m vertices, the intersection with context-free grammar can be performed in time O(m^3) (Bar-Hillel, Perles, & Shamir 1964). The optimal path in such a lattice, according to an n-gram language model, can be found in time O(m) (Cormen *et al.* 2001). In other words, word-lattices meet requirement **B** above.

Although word lattices naturally allow for expressing concatenation and disjunction, they cannot express free/underspecified word order. To represent all possible variations of order for m words, one needs to encode them through different paths in the lattice, which means $m!$ different paths explicitly built in the representation. Word lattices therefore fail to meet requirement **A** of representation compactness.

HALogen uses as means of representing the finite language of a candidate set of realizations the formalism of non-recursive CFGs. This representation is related to word lattices, in the sense that it encodes concatenation and disjunction naturally while still lacking a free-order operator. The advantage of using this representation is that certain sets of substrings, which tend to appear repeatedly, can be replaced by nonterminals that can be used as place-holders for the sets they represent. Although this representation leads to much better compressions of the candidate set than a word lattice, it still suffers from the same drawback, namely, it fails to meet requirement **A**, and one still ends up with an exponentially large representation for such a candidate set (Nederhof & Satta 2004a). In addition, requirement **B** is also space-wise violated

by this representation, as the problem of intersecting a non-recursive CFG with a general CFG is PSPACE-complete (Nederhof & Satta 2004b).

HALogen has been used in end-to-end applications such as machine translation, as a component in generation-heavy systems such as Matador (Habash 2003). The sentence realization stage in Matador creates first a world lattice, which is later converted into a HALogen-compatible forest to be ranked using HALogen's Statistical Forest Ranker (Langkilde 2000). This approach captures translation preferences only at lexical choice level for individual words, and it is not expressive enough to capture phrase-level translation preferences. It also handles word-ordering using a hand-written, ruled-based grammar. As a result, the performance of Matador on translating from Spanish to English falls behind the performance of the IBM4 word-to-word translation model (Brown *et al.* 1993), using a greedy decoder (Germann *et al.* 2001). In comparison, the WIDL-based approach to machine translation taken in this dissertation captures translation preferences for lexical choice at both word and phrase level, and also word-ordering, in a full probabilistic model. The result is state-of-the-art translation performance, measured to exceed the performance of the IBM4 model by a wide margin (Koehn, Och, & Marcu 2003).

Fergus and Amalgam. Other systems, such as Fergus (Bangalore & Rambow 2000b) and Amalgam (Corston-Oliver *et al.* 2002), addressed the syntactic modeling weakness of the n-gram language models used by Nitrogen and HALogen, and leveraged on various syntactic formalisms (such as the Tree-Adjoining Grammar formalism (Joshi 1987) in the case of Fergus)

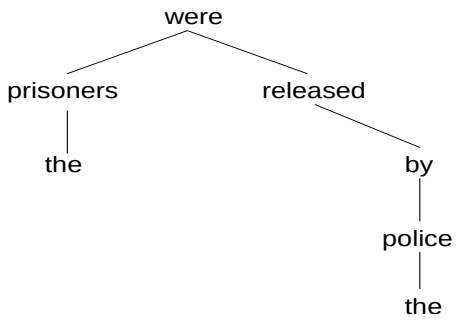

Figure 9.2: Possible Fergus input for the sentence the prisoners were released by the police.

to improve sentence realization. The input taken by systems like Fergus and Amalgam is a set of lexical dependencies, as the example in Figure 9.2 shows.

The representation formalism used by both systems for the set of possible realizations is that of unordered trees. This formalism has the built-in advantage that no order is specified among the children of a given node, and therefore a factorial number of strings is compactly represented in a linear form. Unordered trees are therefore a compact representation that meets requirement **A**. In order to meet requirement **B**, additional constraints are imposed on this representation, such as limiting the number of possible children of a given node. Such constraints make possible efficient algorithms for finding optimal solutions under probabilistic language models of syntax. These systems, therefore, meet both computational requirements outlined in Section 9.1.

Their problems, however, rest in the assumptions built-in at the representation level. From a text-to-text generation perspective, the lexical dependencies assumed as being available for the input representation may be computationally determined when the starting text is an English sentence (or any other language for which a parser is available), but are impossible to determine when parsers are not available, or when the starting text is multiple sentences, or multiple documents.

IDL-expressions. The formalism of IDL-expressions have been proposed by Nederhof and Satta (2004a), but their proposal has not materialized in an actual generation system. The formalism of IDL-expressions, however, has inspired the work described in this dissertation. IDL-expressions are a representation formalism for finite languages, created from strings using four operators: interleave ($\|$), disjunction ($\vee$), lock ($\times$), and precedence ($\cdot$). The precedence ($\cdot$) operator captures precedence constraints, such as the fact that a determiner like the appears before the noun it determines. The lock ($\times$) operator enforces phrase-encoding constraints, such as the fact that the captives is a phrase which should be used as a whole. The disjunction ($\vee$) operator allows for multiple word/phrase choice (e.g., the prisoners versus the captives), and the interleave ($\|$) operator allows for word-order freedom, i.e., word order underspecification at meaning representation level. An example IDL-expression is shown in Figure 9.3.

An important result concerning the representation properties of IDL-expressions states that the size of the IDL representation grows linearly with the size of the input available for generation. This result contrasts with the representational properties of a word lattice or non-recursive CFG representation, which may grow exponentially in n (Nederhof & Satta 2004a). The IDL-expressions formalism, therefore, meets requirement **A**.

The generation mechanism developed for IDL-expressions uses an Earley-style algorithm to intersect IDL-expressions with context-free grammars (CFGs). The output of the generation algorithm is the set of strings that are both in the language represented by the input IDL-expression and accepted by the specified CFG. Although the algorithm is shown to be optimal, from a generation perspective this solution is not feasible, as there can be exponentially many

```
|| (   V ( x(the · prisoners),x(the · captives) )
       V ( were,was )
       V ( x(by · police),x(by · the · police))
       released )
```

Figure 9.3: Possible IDL input for the sentence the prisoners were released by the police.

strings accepted by both formalisms, while one is usually interested only in the most probable realization. The IDL-expressions formalism, therefore, does not meet requirement **B**.

Another important drawback of the IDL formalism is that it cannot handle biases present in the input text. In text-to-text generation, however, such biases are crucial, as they are able to account for information that is present in the input and directly influences the generation process. In Machine Translation, for example, there is almost always the case that a certain foreign word has more than one English correspondent, but there is rarely the case that all the English correspondents are equally likely or appropriate in certain contexts. As such, an operator for choice needs to be able to express these biases. Similarly, an operator for word order needs to be able to express biases as well, for instance the fact that a translation from Chinese to English tends to keep the word order largely monotonic (as both languages are of type SVO), but it also needs to account for certain local movements (e.g., prepositional phrase movement).

9.2 A Computational Perspective on Content Ordering

9.2.1 Content Ordering in Concept-to-Text Generation

The ordering of information has been extensively studied in the context of concept-to-text generation (McKeown 1985; Mooney, Carberry, & McCoy 1990; Dale 1992; Hovy 1993; Bouayad-Agha, Power, & Scott 200). A frequent approach uses a top-down procedure, together with schemas (McKeown 1985) or plans (Dale 1992) to determine the structure and the order of the text to be generated. The underlying assumption is that a rhetorical structure exists that fits the need of the generation task, and this rhetorical structure is known a-priori, or can be selected from a restricted set of predefined structures. The content selection step uses predefined encodings of the propositional content of the input (such as facts extracted from a knowledge database) into the rhetorical elements of the predefined rhetorical structures. The content ordering follows directly from the specification of the rhetorical structure chosen based on the input matches. Text generated using these predefined rhetorical structures exhibits a high degree of coherence and cohesion, due to the precise nature of the structures used. The main disadvantage is that the mechanism is inherently rigid, and difficult to adapt to new requirements or new domains. A possible, less rigid alternative is to combine simple ordering strategies with a set of features extracted from the input (Elhadad & McKeown 2001). In any case, these approaches are all domain dependent, and cannot be made domain-independent, or extended to other domains, in a straightforward manner.

A more flexible mechanism for content ordering uses the Rhetorical Structure Theory (Mann & Thompson 1988) to establish order by means of hierarchical relations between adjacent

propositions. The RST relations are employed using as preconditions constraints based on goals (Hovy 1993) or style (Bouayad-Agha, Power, & Scott 200), and the triggered RST relations determine the order of propositions with respect to other propositions.

A completely different approach to ordering is to use a bottom-up procedure on which proposition order is computed using linguistic notions such as focus (Mooney, Carberry, & McCoy 1990). This is achieved by comparing proposition arguments at a semantic level, which are readily available in context-to-text generation.

9.2.2 Content Ordering in Text-to-Text Generation

Content ordering in text-to-text generation has been addressed using application-specific strategies (Barzilay, Elhadad, & McKeown 2002) or stochastic models of text coherence (Lapata 2003; Barzilay & Lee 2004; Barzilay & Lapata 2005). Ordering strategies are specific to the application for which the generation task is formulated, like multi-document summarization, but are intended to be domain-independent. They use surface indicators to extract features that can be used to implement strategies like Chronological Ordering and Majority Ordering. When the input texts follow similar organizational patters, the Majority Ordering strategy provides good results. When the information presented in the input text is event-based, the Chronological Ordering strategy is effective. If none of the above conditions is true, a combined strategy is shown to be most effective, in which topically-related sentences are group together, and a chronologically-based ordering is inferred for the group sequence (Barzilay, Elhadad, & McKeown 2002).

Ordering text can also be seen as a modeling problem, in which a probabilistic model learns ordering constraints from positive examples. These constraints are afterwards applied to generate text orderings in a stochastic manner. The most widely used constraints are those hypothesized to enforce text coherence, and an increasing number of models of text coherence has been already proposed to date. Lapata's (2003) has been the first probabilistic model of text coherence, and works by learning sequences of lexical and syntactic features that are likely to co-occur. The hypothesis is that phenomena that pertain to local coherence can be captured by these sequences, as they appear in consecutive sentences in the training instances. Other probabilistic model of text coherence include the Content Model of Barzilay and Lee (2004) and the Entity-based Model of Barzilay and Lapata (2005), as well as the text coherence models defined in this dissertation (see Section 6.2 for a detailed description of these models).

Important ingredients for employing stochastic models of text coherence are the search algorithms needed to find the best possible text order among a large number of possible orders. The genetic algorithms of Mellish et al. (1998) and Karamanis and Manarung (2002), as well as the greedy algorithm of Lapata (2003), provide no theoretical guarantees on the optimality of the solutions they propose. At the other end of the spectrum, the exhaustive search of Barzilay and Lee (2004), while ensuring optimal solutions, is prohibitively expensive, and cannot be used to perform utility-based training. The linear programming algorithm of Althaus et al. (2005) is the only proposal that achieves both good speed and accuracy. Their algorithm, however, cannot

handle models with hidden states, cannot compute k-best lists, and does not have the representation flexibility provided by WIDL-expressions, which is crucial for coherence decoding in realistic applications such as multi-document summarization.

Chapter 10

Conclusions

In this dissertation, I formulate and implement a new language generation paradigm, based on direct transformation of textual information into well-formed textual output. In what follows, I articulate the main contributions of this dissertation, as well as new research directions opened by this research.

Theoretical Contributions to Formal Languages and Algorithms. I have defined two formal languages, called WIDL-expressions and WIDL-graphs, and used them as representation formalisms for a multi-purpose generation engine. They are used to represent compactly non-trivial probability distributions over finite, but very large sets of strings, using four operators: precedence ($\cdot$), interleave ($\|$), disjunction ($\vee$), and lock ($\times$). These operators offer enough expressive power to encode state-of-the-art, phrase-based translation models, as shown in Chapter 5. I have also designed and implemented algorithms that use the WIDL formalism to efficiently search, score, and rank strings encoded in WIDL-expressions, in conjunction with

externally-specified probability distributions. These algorithms come with theoretical guarantees about their optimality and efficiency.

However, there are certain theoretical limitations to the expressive power of the WIDL formalism. The representational power of WIDL-graphs is equivalent with that of probabilistic finite-state automata (Chapter 2). Therefore, the set of languages that is possible to encode using WIDL-expressions is a strict subset of the set of (weighted) languages encoded by (probabilistic) tree automata (Rounds 1970). Probabilistic tree automata are a representation framework in which many of the probabilistic tree-based models recently proposed in the literature can be cast (see (Knight & Graehl 2005) for an overview). It is an open question whether the WIDL formalism can be extended such that its representational power matches that of probabilistic tree automata, while maintaining its convenient properties.

Empirical Contributions to Algorithms, Modeling, and State-of-the-art Performance. The empirical evaluations I conducted in this dissertation allow one to empirically validate the advantages, measured in time and space savings, of using WIDL-based algorithms for language generation. They also allow one to empirically validate the discriminative power of various stochastic models of language for the task of language generation. At sentence level, I have shown that a syntax-based language model, based on a generative model of syntax, improves over the performance of an n-gram language model alone, on a word reordering task, when short sentences are considered (Chapter 8). For longer sentences, the performance does not improve significantly, and an error analysis shows that the syntax-based language model is still weak in enforcing correctness globally, when dealing with a vast number of possible realizations. At

discourse level, the WIDL framework allows one to utility-train a combination of stochastic coherence models, and the result is a log-linear coherence model with greater discriminative power than previously-proposed coherence models (Chapter 7).

Last but not least, these evaluations allow one to compare the overall performance against state-of-the-art performance on headline generation, machine translation, and coherent discourse generation. They show state-of-the-art performance in automatic translation (Chapter 5), and significant improvements in state-of-the-art performance in abstractive headline generation (Chapter 4) and coherent discourse generation (Chapter 7).

The performance of these applications, however, is directly influenced by the ability of the stochastic language models employed to differentiate between correct and incorrect language usage. There has been a lot of progress in modeling formal languages since the 1957 famous Chomskian problem to distinguish between grammatical and non-grammatical English (Chomsky 1957). On the other hand, there has been much less progress in modeling natural languages, to the extent that we still do not have a high-accuracy method to solve Chomsky's above-stated problem, much less the problem of distinguishing between sensible and non-sensible English. A renewed focus on language modeling, boosted by the generation paradigm proposed in this dissertation, might contribute towards solving these basic, but still open, natural language problems.

Reference List

[Alshawi, Bangalore, & Douglas 2000] Alshawi, H.; Bangalore, S.; and Douglas, S. 2000. Learning dependency translation models as collections of finite-state head transducers. *Computational Linguistics* 26(1):45–60.

[Althaus, Karamanis, & Koller 2005] Althaus, E.; Karamanis, N.; and Koller, A. 2005. Computing locally coherent discourse. In *Proceedings of the ACL*, 399–406.

[Bangalore & Rambow 2000a] Bangalore, S., and Rambow, O. 2000a. Corpus-based lexical choice in natural language generation. In *Proceedings of the Association of Computational Linguistics (ACL 2000)*.

[Bangalore & Rambow 2000b] Bangalore, S., and Rambow, O. 2000b. Using TAG, a tree model, and a language model for generation. In *Proceedings of the Fifth International Workshop on Tree-Adjoining Grammars (TAG+)*.

[Banko, Mittal, & Witbrock 2000] Banko, M.; Mittal, V.; and Witbrock, M. 2000. Headline generation based on statistical translation. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL–2000)*, 318–325.

[Bar-Hillel, Perles, & Shamir 1964] Bar-Hillel, Y.; Perles, M.; and Shamir, E. 1964. *On formal properties of simple phrase structure grammars*. Addison-Wesley. chapter 9, 116–150.

[Barzilay & Lapata 2005] Barzilay, R., and Lapata, M. 2005. Modeling local coherence: An entity-based approach. In *Proceedings of the ACL*, 141–148.

[Barzilay & Lee 2004] Barzilay, R., and Lee, L. 2004. Catching the drift: Probabilistic content models, with applications to generation and summarization. In *Proceedings of the HLT-NAACL*, 113–120.

[Barzilay, Elhadad, & McKeown 2002] Barzilay, R.; Elhadad, N.; and McKeown, K. R. 2002. Inferring strategies for sentence ordering in multidocument news summarization. *Journal of Artifi cial Intelligence Research* 17:35–55.

[Barzilay 2003] Barzilay, R. 2003. *Information Fusion for Multidocument Summarization: Paraphrasing and Generation*. Ph.D. Dissertation, Columbia University.

[Bergler *et al.* 2003] Bergler, S.; Witte, R.; Khalife, M.; Li, Z.; and Rudzicz, F. 2003. Using knowledge-poor coreference resolution for text summarization. In *Proceedings of the Document Understanding Conference (DUC 2003)*.

[Bikel 2004] Bikel, D. M. 2004. Intricacies of Collins' parsing model. *Computational Linguistics*.

[Bouayad-Agha, Power, & Scott 200] Bouayad-Agha, N.; Power, R.; and Scott, D. 200. Can text structure be incompatible with rhetorical structure? In *Proceedings of the First International Conference on Natural Language Generation (INLG-2000)*.

[Brown *et al.* 1993] Brown, P.; Della Pietra, S.; Della Pietra, V.; and Mercer, R. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics* 19(2):263–311.

[Canning *et al.* 2000] Canning, Y.; Tait, J.; Archibald, J.; and Crawley, R. 2000. Cohesive generation of syntatically simplified newspaper text. In *Proceedings of the Workshop on Robust Methods in Analysis of Natural Language Data*, 145–150.

[Carlson, Marcu, & Okurowski 2003] Carlson, L.; Marcu, D.; and Okurowski, M. E. 2003. Building a discourse-tagged corpus in the framework of Rhetorical Structure Theory. In van Kuppevelt, J., and Smith, R., eds., *Current Directions in Discourse and Dialogue*. Kluwer Academic Publishers. To appear.

[Chandrasekar, Doran, & Bangalore 1996] Chandrasekar, R.; Doran, C.; and Bangalore, S. 1996. Motivations and methods for text simplification. In *Proceedings of the Sixteenth International Conference on Computational Linguistics (COLING '96)*.

[Charniak, Knight, & Yamada 2003] Charniak, E.; Knight, K.; and Yamada, K. 2003. Syntax-based language models for machine translation. In *Proceedings of the MT Summit IX*.

[Charniak 2000] Charniak, E. 2000. A maximum-entropy-inspired parser. In *Proceedings of the NAACL 2000*, 132–139.

[Charniak 2001] Charniak, E. 2001. Immediate-head parsing for language models. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*.

[Chelba & Jelinek 1998] Chelba, C., and Jelinek, F. 1998. Exploiting syntactic structure for language modeling. In *Proceedings of International Committee on Computational Linguistics/Conference of the Association for Computational Linguistics*.

[Chelba 2000] Chelba, C. 2000. *Exploiting Syntactic Structure for Natural Language Modeling*. Ph.D. Dissertation, Johns Hopkins University.

[Chiang 2005] Chiang, D. 2005. A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the Association of Computational Linguistics (ACL)*, 263–270.

[Chomsky 1957] Chomsky, N. 1957. *Syntactic structures*. The Hague: Mouton & Co.

[Collins 1999] Collins, M. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. Dissertation, University of Pennsylvania.

[Collins 2003] Collins, M. 2003. Head-driven statistical models for natural language parsing. *Computational Linguistics*.

[Cormen *et al.* 2001] Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; and Stein, C. 2001. *Introduction to Algorithms*. The MIT Press and McGraw-Hill. Second Edition.

[Corston-Oliver *et al.* 2002] Corston-Oliver, S.; Gamon, M.; Ringger, E. K.; and Moore, R. 2002. An overview of amalgam: A machine-learned generation module. In *Proceedings of the International Natural Language Generation Conference*.

[Dale 1992] Dale, R. 1992. *Generating Referring Expressions: Constructing Descriptions in a Domain of Objects and Processes*. Cambridge, Massachusetts: MIT Press.

[Daume *et al.* 2002] Daume, H.; Knight, K.; Langkilde-Geary, I.; Marcu, D.; and Yamada, K. 2002. The importance of lexicalized syntax models for natural language generation tasks. In *Proceedings of the Second International Conference on Natural Language Generation*.

[Dorr, Zajic, & Schwartz 2003] Dorr, B.; Zajic, D.; and Schwartz, R. 2003. Hedge trimmer: a parse-and-trim approach to headline generation. In *Proceedings of the HLT-NAACL Text Summarization Workshop*, 1–8.

[Elhadad & McKeown 2001] Elhadad, N., and McKeown, K. R. 2001. Generating patient specific summaries of medical articles. In *Proceedings of the NAACL 2001 Workshop on Automatic Summarization*.

[Elhadad 1991] Elhadad, M. 1991. FUF User manual — version 5.0. Technical Report CUCS-038-91, Department of Computer Science, Columbia University.

[Forbes *et al.* 2001] Forbes, K.; Miltsakaki, E.; Prasad, R.; Sarkar, A.; Joshi, A.; and Webber, B. 2001. D-LTAG System: Discourse parsing with a lexicalized tree-adjoining grammar. In *ESSLLI'2001 Workshop on Information Structure, Discourse Structure and Discourse Semantics*.

[Galley *et al.* 2004] Galley, M.; Hopkins, M.; Knight, K.; and Marcu, D. 2004. What's in a translation rule? In *In Proceedings of the Human Language Technology and North American Association for Computational Linguistics Conference (HLT/NAACL 2004)*.

[Germann *et al.* 2001] Germann, U.; Jahr, M.; Knight, K.; Marcu, D.; and Yamada, K. 2001. Fast decoding and optimal decoding for machine translation. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL'01)*, 228–235.

[Goodman 2001] Goodman, J. 2001. A bit of progress in language modeling. Technical Report MSR-TR-2001-72, Microsoft Research.

[Grosz, Joshi, & Weinstein 1995] Grosz, B.; Joshi, A.; and Weinstein, S. 1995. Centering: A framework for modeling the local coherence of discourse. *Computational Linguistics* 21(2):203–226.

[Habash 2003] Habash, N. 2003. Matador: A large-scale Spanish-English GHMT system. In *Proceedings of the Conference of the Association for Machine Translation in the Americas (AMTA)*.

[Habash 2004] Habash, N. 2004. The use of a structural n-gram language model in generation-heavy hybrid machine translation. In *Proceedings of the International Conference on Natural Language Generation (INLG 2004)*.

[Haegeman 1994] Haegeman, L. 1994. *Introduction to Government and Binding Theory*. Blackwell Publishers, second edition edition.

[Hajic *et al.* 2002] Hajic, J.; Cmejrek, M.; Dorr, B.; Ding, Y.; Eisner, J.; Gildea, D.; Koo, T.; Parton, K.; Penn, G.; Radev, D.; and Rambow, O. 2002. Natural language generation in the context of machine translation. Summer workshop final report, Johns Hopkins University.

[Hovy & Lin 2000] Hovy, E., and Lin, C. 2000. Automated text summarization in summarist. In Mani, I., and Maybury, M., eds., *Advances in Automatic Text Summarization*. The MIT Press. 81–94.

[Hovy 1993] Hovy, E. 1993. Automated discourse generation using discourse structure relations. *Artifi cial Intelligence* 63(1–2):341–386.

[Huang & Chiang 2005] Huang, L., and Chiang, D. 2005. Better k-best parsing. In *Proceedings of the 9th International Workshop on Parsing Technologies (IWPT 2005)*.

[Jing & McKeown 1999] Jing, H., and McKeown, K. 1999. The decomposition of human-written summary sentences. In *Proceedings of the 22nd Conference on Research and Development in Information Retrieval (SIGIR–99)*.

[Joshi 1987] Joshi, A. 1987. An introduction to tree adjoining grammar. In Manaster-Ramer, A., ed., *Mathematics of Language*. John Benjamins.

[Karamanis & Manurung 2002] Karamanis, N., and Manurung, H. M. 2002. Stochastic text structuring using the principle of continuity. In *Proceedings of INLG*, 81–88.

[Karamanis *et al.* 2004] Karamanis, N.; Poesio, M.; Mellish, C.; and Oberlander, J. 2004. Evaluating centering-based metrics of coherence for text structuring using a reliably annotated corpus. In *Proceedings of the ACL*.

[Kibble & Power 2004] Kibble, R., and Power, R. 2004. Optimising referential coherence in text generation. *Computational Linguistics* 30(4):410–416.

[Klein & Manning 2003] Klein, D., and Manning, C. 2003. A* parsing: Fast exact viterbi parse selection. In *Proceedings of the Human Language Technology and North American Association for Computational Linguistics Conference (HLT/NAACL-03)*.

[Knight & Graehl 1998] Knight, K., and Graehl, J. 1998. Machine transliteration. *Computational Linguistics* 24(4):599–612.

[Knight & Graehl 2005] Knight, K., and Graehl, J. 2005. An overview of probabilistic tree transducers for natural language processing. In *In Proceedings of the Sixth Conference on Intelligent Text Processing and Computational Linguistics (CICLing)*.

[Knight & Hatzivassiloglou 1995] Knight, K., and Hatzivassiloglou, V. 1995. Two level, many-path generation. In *Proceedings of the Association of Computational Linguistics*.

[Knight & Marcu 2002] Knight, K., and Marcu, D. 2002. Statistics-based summarization — step one: Sentence compression. *Artifi cial Intelligence* 139(1).

[Knight 1999] Knight, K. 1999. Decoding complexity in word-replacement translation models. *Computational Linguistics* 25(4).

[Koehn, Och, & Marcu 2003] Koehn, P.; Och, F. J.; and Marcu, D. 2003. Statistical phrase based translation. In *Proceedings of the Joint Conference on Human Language Technologies and the Annual Meeting of the North American Chapter of the Association of Computational Linguistics (HLT-NAACL)*, 127–133.

[Koehn 2004] Koehn, P. 2004. Pharaoh: a beam search decoder for phrase-based statistical machine translтion models. In *Proceedings of the Sixth Conference of the Association for Machine Translation in the Americas*, 115–124.

[Langkilde-Geary 2002] Langkilde-Geary, I. 2002. *A foundation for general-purpose natural language generation: sentence realization using probabilistic models of language*. Ph.D. Dissertation, University of Southern California.

[Langkilde 2000] Langkilde, I. 2000. Forest-based statistical sentence generation. In *Proceedings of the 1st Annual Meeting of the North American Chapter of the Association for Computational Linguistics*.

[Lapata 2003] Lapata, M. 2003. Probabilistic text structuring: Experiments with text ordering. In *Proceedings of the ACL*, 545–552.

[Lin 2004] Lin, C.-Y. 2004. ROUGE: a package for automatic evaluation of summaries. In *Proceedings of the Workshop on Text Summarization Branches Out (WAS 2004)*.

[Mann & Thompson 1988] Mann, W., and Thompson, S. 1988. Rhetorical Structure Theory: Toward a functional theory of text organization. *Text* 8(3):243–281.

[Marcu 2000] Marcu, D. 2000. *The Theory and Practice of Discourse Parsing and Summarization*. Cambridge, Massachusetts: The MIT Press.

[Marcus, Santorini, & Marcinkiewicz 1993] Marcus, M.; Santorini, B.; and Marcinkiewicz, M. 1993. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics* 19(2):313–330.

[Matthiessen & Bateman 1991] Matthiessen, C., and Bateman, J. 1991. *Text Generation and Systemic-Functional Linguistic*. London: Pinter Publishers.

[McKeown 1985] McKeown, K. 1985. *Text Generation: Using Discourse Strategies and Focus Constraints to Generate Natural Language Text*. Cambridge University Press.

[Mellish *et al.* 1998] Mellish, C.; Knott, A.; Oberlander, J.; and O'Donnell, M. 1998. Experiments using stochastic search for text planning. In *Proceedings of the 9th International Workshop on Natural Language Generation*, 98–107.

[Meteer *et al.* 1987] Meteer, M.; McDonald, D.; Anderson, S.; Foster, D.; Gay, L.; Iluettner, A.; and Sibun, P. 1987. MUMBLE-86: Design and implementation. Technical Report COINS 87-87, University of Massachusets, Amherst, Amherst, MA.

[Mohri, Pereira, & Riley 2002] Mohri, M.; Pereira, F.; and Riley, M. 2002. Weighted finite-state transducers in speech recognition. *Computer Speech and Language* 16(1):69–88.

[Mooney, Carberry, & McCoy 1990] Mooney, D.; Carberry, S.; and McCoy, K. 1990. The generation of high-level structure for extended explanations. In *Proceedings of the International Conference on Computational Linguistics (COLING–90)*, volume 2, 276–281.

[Morris & Hirst 1991] Morris, J., and Hirst, G. 1991. Lexical cohesion computed by thesaural relations as an indicator of the structure of text. *Computational Linguistics* 17(1):21–48.

[Nederhof & Satta 2004a] Nederhof, M.-J., and Satta, G. 2004a. IDL-expressions: a formalism for representing and parsing finite languages in natural language processing. *Journal of Artifi cial Intelligence Research* 287–317.

[Nederhof & Satta 2004b] Nederhof, M.-J., and Satta, G. 2004b. The language intersection problem for non-recursive context-free grammars. *Information and Computation*.

[Och & Ney 2002] Och, F., and Ney, H. 2002. Discriminative training and maximum entropy models for statistical machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, 295–302.

[Och *et al.* 2004] Och, F. J.; Gildea, D.; Khudanpur, S.; Sarkar, A.; Yamada, K.; Fraser, A.; Kumar, S.; Shen, L.; Smith, D.; Eng, K.; Jain, V.; Jin, Z.; and Radev, D. 2004. A smorgasbord of features for statistical machine translation. In *Proceedings of the Human Language Technology and North American Association for Computational Linguistics Conference (HLT/NAACL-04)*.

[Och, Tillmann, & Ney 1999] Och, F.; Tillmann, C.; and Ney, H. 1999. Improved alignment models for statistical machine translation. In *Proceedings of the Joint Workshop on Empirical Methods in NLP and Very Large Corpora*, 20–28.

[Och 2003] Och, F. 2003. Minimum error rate training in statistical machine translation. In *Proceedings of the ACL*, 160–167.

[Papineni *et al.* 2002] Papineni, K.; Roukos, S.; Ward, T.; and Zhu, W.-J. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL-2002)*, 311–318.

[Papineni, Roukos, & Ward 1997] Papineni, K.; Roukos, S.; and Ward, T. 1997. Feature-based language understanding. In *Proceedings of European Conference on Speech Communication and Technology*, 1435–1438.

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

[Rounds 1970] Rounds, W. C. 1970. Mappings and grammars on trees. *Mathematical Systems Theory* 4:257–287.

[Russell & Norvig 1995] Russell, S., and Norvig, P. 1995. *Artifi cial Intelligence. A Modern Approach*. Prentice Hall, Englewood Cliffs, New Jersey.

[Schafer & Yarowsky 2003] Schafer, C., and Yarowsky, D. 2003. Statistical machine translation using coercive two-level syntactic transduction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

[Schwartz 2001] Schwartz, R. 2001. Unsupervised topic discovery. In *Proceedings of Workshop on Language Modeling and Information Retrieval, pp.72-77*, 72–77.

[Scott & de Souza 1990] Scott, D., and de Souza, C. 1990. Getting the message across in RST-based text generation. In Dale, R.; Mellish, C.; and Zock, M., eds., *Current Research in Natural Language Generation*. New York: Academic Press. 47–73.

[Soricut & Brill 2004] Soricut, R., and Brill, E. 2004. Automatic question answering: Beyond the factoid. In *Proceedings of the Human Language Technology and North American Association for Computational Linguistics Conference*.

[Stolcke 2002] Stolcke, A. 2002. Srilm - an extensible language modeling toolkit. In *Proceedings of the International Conference on Spoken Language Processing*.

[Wan *et al.* 2005] Wan, S.; Dale, R.; Dras, M.; and Paris, C. 2005. Statistically generated summary sentences: A preliminary evaluation using a dependency relation precision metric. In *Proceedings of the Workshop on Using Corpora for Natural Language Generaton*.

[Wu 1997] Wu, D. 1997. Stochastic inversion transduction grammars and bilingual parsing for parallel corpora. *Computational Linguistics* 23(3).

[Yamada & Knight 2001] Yamada, K., and Knight, K. 2001. A syntax-based statistical translation model. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL'01)*.

[Zajic, Dorr, & Schwartz 2004] Zajic, D.; Dorr, B. J.; and Schwartz, R. 2004. BBN/UMD at DUC-2004: Topiary. In *Proceedings of the North American Chapter of the Association for Computational Linguistics Workshop on Document Understanding*, 112–119.

[Zhou & Hovy 2003] Zhou, L., and Hovy, E. 2003. Headline summarization at ISI. In *Proceedings of the Document Understanding Conference (DUC 2003)*.