VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

FINITE STATE GRAMMARS AS LANGUAGE MODELS FOR AUTOMATIC SPEECH RECOGNITION

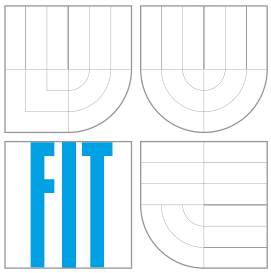
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

KAREL BENEŠ

BRNO 2014

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

KONEČNĚ STAVOVÉ GRAMATIKY JAKO JAZYKOVÉ MODELY PRO AUTOMATICKÝ PŘEPIS ŘEČI

FINITE STATE GRAMMARS AS LANGUAGE MODELS FOR AUTOMATIC SPEECH RECOGNITION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

KAREL BENEŠ

VEDOUCÍ PRÁCE
SUPERVISOR

Dipl.-Ing. MIRKO HANNEMANN

BRNO 2014

Abstrakt

Tato práce se zabývá transformací bezkontextových gramatik na váhované konečně stavové převodníky. Je vybrána podmnožina bezkontextových gramatik, kterou lze tranformovat přesně. Je představen test, zda daná gramatika naleží do této podmnožiny, i algoritmus převodu. Dále je popsán vlastní nástroj, který tyto postupy implementuje, včetně způsobu zpracování vstupu a výstupu. S použitím toho nástroje byl vytvořen systém rozpoznání řeči pro kokpit letadla. Jsou představeny výsledky ukazující, že systém založený na takto získaném modelu jazyka podává výrazně lepší výkon, než je dosažen při použití obecného modelu.

Abstract

This thesis deals with the transformation of Context Free Grammars (CFG) into Weighted Finite State Transducers (WFST). A subset of CFG is chosen, that can be transformed exactly. Both the test of whether a CFG fulfills such condition and the algorithm for the following transformation are presented. A tool has been implemented, which performs both these tasks, also its input and output processing are reported. Using this tool, a speech recognition system for aircraft cockpit control has been built. Results are presented which show, that the system based on the transformed grammar outperforms the system based on general-purpose language model.

Klíčová slova

jazykový model, bezkontextová gramatika, váhované konečně stavové převodníky, rozpoznávací síť, automatické rozpoznávání řeči

Keywords

Language Model, Context Free Grammar, Weighted Finite State Transducer, recognition network, automatic speech recognition

Citace

Karel Beneš: Finite State Grammars as Language Models for Automatic Speech Recognition, bakalářská práce, Brno, FIT VUT v Brně, 2014

Finite State Grammars as Language Models for Automatic Speech Recognition

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Dipl.-Ing. Mirko Hannemanna. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal. Systémy popsané ve čtvrté kapitole jsou založeny na práci dalších členů skupiny Speech@FIT, toto je vždy explicitně uvedeno.

. . . . . .
Karel Beneš
May 19, 2014

Poděkování

I would like to thank my supervisor Mirko Hannemann for his patience, guidance and support. I am grateful to him for the thorough constructive criticism on this thesis.

I would like to thank Kateřina Žmolíková for the advice on the use of the system, which she has trained.

Contents

Chapter 1

Introduction

Speech recognition nowadays is based on a statistical approach, where the parameters of the models are trained on huge amounts of data, with humans directly affecting only the structure of these models. However, the situation may arise, that additional constraints on the input speech are known. This is usually the case when recognizing domain- or even application-specific speech. In this case, we can exploit them to improve the performance of our systems.

This thesis deals with such a situation and presents a way to use the knowledge of the typical utterances spoken, when we are given a context free grammar (CFG) describing it.

At first, the key concepts of speech recognition are introduced in chapter 2, including the mathematical definition of automatic speech recognition (ASR), an intuition for the process of ASR and the error metrics used in the evaluation of the experiments.

Chapter 3 describes the design of a tool developed for transformation of the input knowledge encoded as a CFG into a Weighted Finite State Transducer (WFST) used in modern speech recognition systems. Details about design decisions limiting the set of accepted input grammars are presented, as well as nontrivial input transformations, that allow the tool to work with a real-world grammar specification.

Finally, in chapter 4, the application of this tool in a project, in which the Speech@FIT group participates, is described. Specific issues of the provided grammars are shown and a basic description of the system is given. Then, taking an example from this project, experiments are described and their result are discussed.

Chapter 2

Speech recognition with Weighted Finite State Transducers

Speech recognition is a complex process, that can be, from both the engineering and theoretical point of view, decomposed into several independent parts. The basic structure is captured in figure 2.1. The signal processing computes features from the input audio signal. The acoustic modeling then evaluates to which class (e.g. silence, particular phonemes, noise etc.) the sound most likely belongs. During the decoding, this information is combined with a model of language, which constrains the possible sequences of words, to give the most likely word sequence as recognition output.

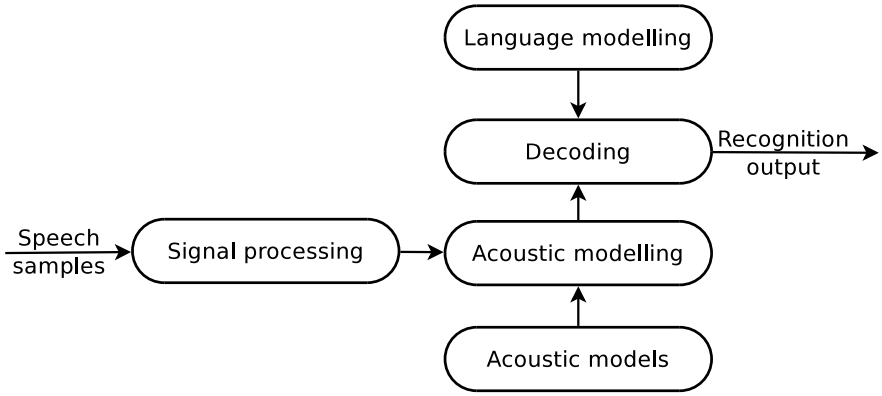


Figure 2.1: Simple schema of a speech recognizer. This work deals mainly with the language modelling and decoding parts.

This work deals mainly with language modeling, with respect to the decoding, thus a popular model for representing language constraints – the Weighted Finite State Transducer – is presented in this chapter, as well as a statistical tool – the Hidden Markov Model – that connects it to acoustic modeling, so that the speech recognition is theoretically well defined.

2.1 Hidden Markov Models as models for sequence matching

The Hidden Markov Model (HMM) is a statistical model typically used for modeling temporal sequences. It models a system in which the distribution of observed variables is

determined by a state, which itself is an unobserved process. Following [13], the HMM can be defined as a 5-tuple:

$$\text{HMM} = (N, M, A, B, \pi)$$

where: N denotes the number of states. M defines the output space ([13] defines it as discrete, in speech recognition the whole continuous space of required dimensionality is usually considered). The transition probability distribution $A = \{A_{ij}\}$ describes the probability of moving from a given state to another in a single step, with $A_{ij} = P(S_{n+1} = j | S_n = i)$. B is a probability density function (probability mass function for discrete M) of observed variables given the actual state of the HMM. We can write it as $B = b_i(\mathbf{x})$, where $\mathbf{x}$ is a vector from the sample space. Finally, π is a probability mass function, where π_i is the probability of state i being the initial state. From the probabilistic point of view, HMMs are well described as graphical models, as captured in figure 2.2.

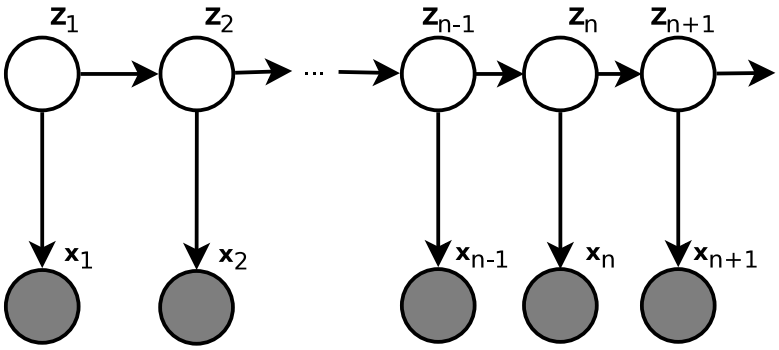


Figure 2.2: The Hidden Markov Model shown as a graphical model. This schema has been taken from [7]. This is a model of an observed sequence of arbitrary length. The shaded nodes $\mathbf{x}_i$ correspond to the distributions of the observed samples, while the uncolored nodes $\mathbf{z}_i$ correspond to the hidden Markov process. Each arrow symbolizes conditioning, e.g. the arrow from $\mathbf{z}_1$ to $\mathbf{x}_1$ implies, that we know the distribution $p(\mathbf{x}_1|\mathbf{z}_1)$ or that the distribution of $\mathbf{x}_1$ is given in terms of $\mathbf{z}_1$. Also the Markov property is captured in the fact, that each $\mathbf{z}_{i+1}$ is conditioned on $\mathbf{z}_i$ only.

The HMM can be viewed as a generative model: Assign a token to a random initial state i according to π. Then draw a sample from the corresponding probability density function b_i. Finally choose the next state j according to A. Then loop over the steps 'draw a sample' and 'move the token' for the desired number of times. There is no notion of a 'final state' in the HMM, so the token does not need to reach it.

In terms of HMM, the task of speech recognition can be described as finding the most probable sequence S' of the hidden states, representing phonetic (sub)units, e.g. phones or triphones, given X, the observed sequence of features. This process is generally referred to as *decoding*. This can be expressed using Bayes' theorem:

$$S' = \arg\max_S \; P(S|X) = \arg\max_S \; \frac{P(X|S)P(S)}{P(X)}$$

Since the denominator is a constant with respect to S, it can be omitted:

$$S' = \arg\max_S \; \{P(X|S)P(S)\}$$

This factorization directly relates to the process of speech recognition. The $P(S)$ is the prior probability of the sequence. This prior probability reflects the pronunciation of words in the language and the expected word order. From a formal point of view, this knowledge is captured in the transition probability distribution A. It is usually referred to as the *language model*[1]. Actual techniques used for language modelling are introduced in 2.2. On the other hand, the likelihood of the observed sequence $P(X|S)$ relates the phones (hidden states of the HMM) to the actual observed samples. This is referred to as the *acoustic model*.

Decoding may be viewed, and is often implemented, as passing several tokens through the HMM, where every token has a probability associated with it. At every decoding step n, for each transition of non-zero probability leading from a state S_n where there is a token t, a new token s is generated in the state S_{n+1}. Let $\mathbf{x}_n$ denote the n-th observed sample. Then the likelihood of the token s can be described as $P(s) = P(t)A_{S_nS_{n+1}}b_{S_{n+1}}(\mathbf{x}_{n+1})$. Out of all the tokens obtained this way, only a certain number of those with the highest likelihood is kept for further processing. This is usually referred to as the *width of the decoding beam*.

The estimation of the parameters of an HMM is a non-trivial task which is of no direct impact on this thesis, thus it will not be further discussed here – [13] presents the classical Baum-Welch algorithm. Generally the training follows the Expectation-Maximization scheme [7].

2.2 Weighted Finite State Transducers

Several components (pronunciation model, language model) of modern speech recognizers can be naturally described by some form of finite automaton. As a common framework for formally defining these, weighted finite state transducers are used. The following paragraphs are based on [15].

Being an extension of finite state automata (FSA), weighted finite state transducers (WFST) can be defined as an 8-tuple:

$$T = (Q, \Sigma, \Gamma, I, F, E, \lambda, \rho)$$

where:

Q	is a finite set of states
Σ	is a finite input alphabet
Γ	is a finite output alphabet
$I \subseteq Q$	is the set of initial states
$F \subseteq Q$	is the set of final states
$E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\}) \times Q \times K$	is the set of transitions
$\lambda : I \to \mathbb{K}$	defines weights of initial states
$\rho : F \to \mathbb{K}$	defines weights of final states

where $\mathbb{K}$, the set of weights, forms a suitable semiring[2]. Therefore, the WFST can be seen as a finite automaton, where each transition has an output symbol and a weight associated. Thus the basic idea of how a WFST operates on an input string is the same as with the finite state automaton.

The usage of the semiring $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ is the following: Let a *path* $\pi = e_1 \cdots e_k$ be a sequence of consecutive transitions, that is for $i = 1, \ldots, k-1$ the destination state of e_i

[1] The term *language model* has two meanings in speech recognition. Here it is applied in broader terms containing pronunciation dictionary as well, the other meaning puts constraints on the word level only.

[2] This is not a mandatory condition, but is necessary for practical applications.

is equal to the origin of e_{i+1}. Its weight is defined as the $\otimes$-product of the weights of the single transitions: $w[\pi] = w[e_1] \otimes \cdots \otimes w[e_k]$. The weight of a finite set of paths R is defined as $w[R] = \bigoplus_{\pi \in R} w[\pi]$. Depending on the properties of the semiring, this may be defined for infinite sets of paths as well.

Similar to classical FSA, operations such as *weighted determinization* and *minimization* are defined on WFSTs. The exact algorithms will not be discussed here as they are not of any significant importance for this work. Yet it is noteworthy that a WFST is not generally determinizable, depending on the properties of the underlying semiring K and satisfaction of the *twins property* [8]. These operations, especially determinization, have the usual great impact on the practical usage of a WFST.

However, the fact that a WFST translates an input string to an output one, introduces another important operation: *composition*. A WFST $C = A \circ B$, composed of WFSTs A and B, translates strings over Σ_A into strings over Γ_B in a way equivalent to first A operating on the input string and then B operating on the output of A. Another way to view this, is to see a WFST as a binary relation between strings. Then the relation composition arises as a natural operation. However, this view is informal, because it does not take the weights into account.

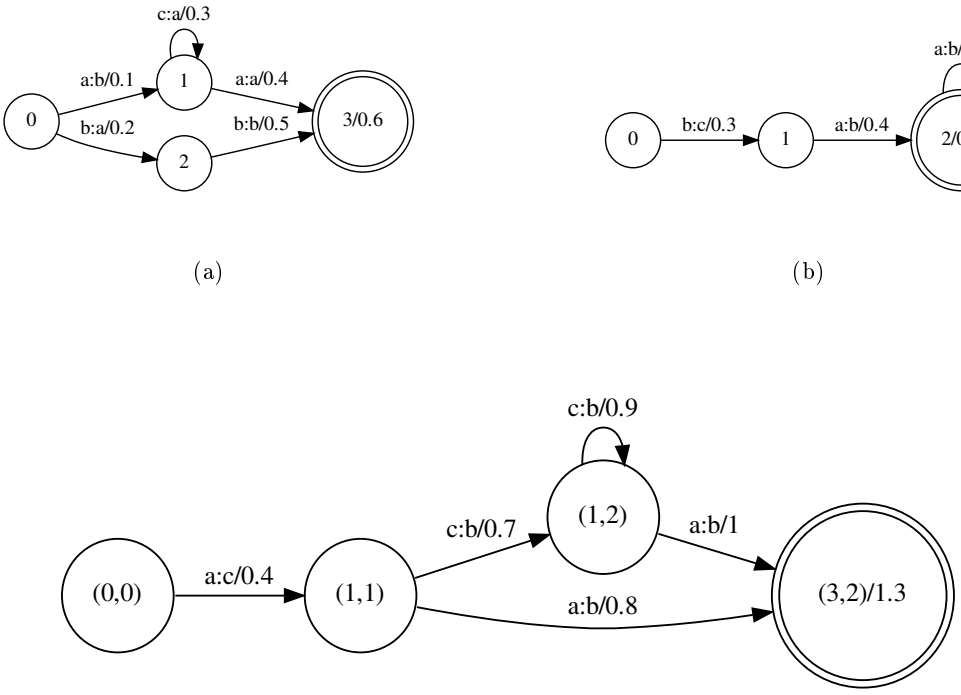


Figure 2.3: Example of WFST composition, where the $\otimes$ operation is equivalent to sum (as with log probabilities). The transducer 2.3c is composition of 2.3a and 2.3b. Node caption of each state in the transducer 2.3c refers to the pair of states of the original transducers. Example taken from [15]

The algorithm of composition is similar to constructing the intersection of FSAs. For

$C = A \circ B$, a state of C represents a pair of an A state and a B state. A state of C is initial, exactly when the corresponding states of A and B are both initial, and it is a final state when the corresponding states of A and B are both final. For every pair of transitions $q_1 \to q_2$ in A and $r_1 \to r_2$ in B, such that the output symbol of $q_1 \to q_2$ matches the input symbol of $r_1 \to r_2$, there is a transition $(q_1, r_1) \to (q_2, r_2)$ in C. The weight of this transition is computed as $w[q_1 \to q_2] \otimes w[r_1 \to r_2]$. The weights of final states are the $\otimes$-products of the weights of the corresponding final states in A and B.

An example of the WFST composition is given in figure 2.3.

2.3 Creating a recognition network

Phones are not independent of each other in speech. They physically affect the preceding and following phones, this effect is known as *coarticulation* [11], and from all possible sequences of phones, only a small portion forms semantically valid units. In order to enhance speech recognition accuracy, both issues must be addressed.

The former one is typically handled by considering context dependent phones, typically triphones, as the target of acoustic modelling. The HMM parameters can then be estimated to capture most of the co-articulation effect. To solve the latter issue, knowledge of the spoken language (pronunciation, grammar, etc.) is needed. Using the WFST framework, this knowledge is encoded as a WFST. Considering the unobserved part (the hidden states) of the HMM to be a WFST as well, WFST composition can be used to put all these components together. The result can then be thought of as a single big HMM, called *recognition network*. The three typical components are the language model, the pronunciation dictionary and the context dependency transducer.

The language model captures constraints put on words sequences. These constraints may be given in form of a hand-crafted grammar or a statistical n-gram model. Grammars are typical for domain-specific speech recognition, while statistical models are used for general purpose speech recognition.

The statistical n-gram models are based on the Markov assumption, that the conditional distribution of a word at a given position is dependent only on the $n-1$ previous words. The n-gram models can be approximated as a WFST in a straight forward manner, where each state represents a particular history, though their spatial complexity grows exponentially with n. Sophisticated ways of pruning the n-gram models were developed with some form of finite state machine as the target, so there is no extra effort needed to put them into work. An example of bigram (2-gram) model is given in figure 2.4.

Regular and linear grammars can also be easily coded as WFSTs, and as will be shown in chapter 3, a useful subset of context free grammars (CFG) as well. Furthermore, several methods of approximation of arbitrary CFG have been developed (see [17]). This WFST is usually referred to as G (for grammar) and its weights play an important role in decoding, especially with statistical models.

There has also been a successful attempt to model probabilities of word sequences with recurrent neural networks [16], however, this is a very different technique and will not be discussed further.

The pronunciation dictionary captures possible pronunciations of words, that is which phone sequences form a given word. It is often referred to as *lexicon*. A typical lexicon is

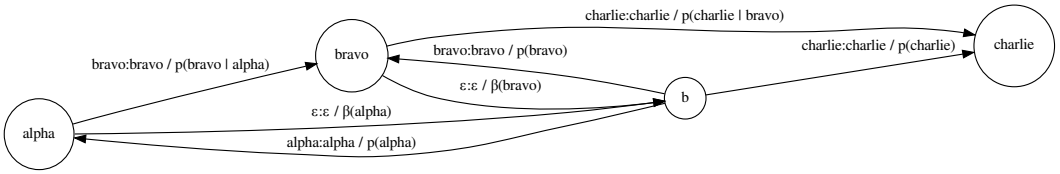


Figure 2.4: Example of WFST corresponding to bigram a model. Each state represents a history (one word for bigram), the 'b' state is a *backoff* state. It represents no history and is used for modeling probabilities of sequences which could not be reasonably trained, because they were not seen in the training data or were too rare. In this example, the word sequence "alpha charlie" was not seen often enough to estimate its probability, so it is modeled using this state.

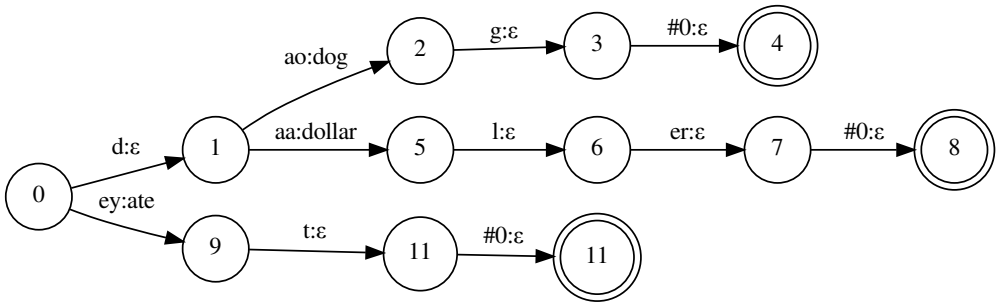


Figure 2.5: A simple lexicon representing the pronunciation of words 'dog', 'dollar' and 'ate'.

represented as a tree structured WFST, as illustrated in figure 2.5. Note however, that it is generally not determinizable because of homophones[3]. To solve this problem, disambiguation symbols, typically of the form $\#_k$ are introduced at the end of each word, so that each phone sequence is unique. Also it may happen, that word boundaries would not be clear in the phone sequences, so these disambiguation symbols are appended to all words. These symbols are removed later in the process of creating the decoding graph. This WFST is usually referred to as L (for lexicon). Weights may be used to capture probabilities of different pronunciation variants, otherwise those equal to $\overline{1}$ are used.

The context dependency transducer is introduced for implementation purposes: As mentioned earlier, the acoustic modeling HMM is usually trained to classify context dependent phonemes, whereas the lexicon translates words into context independent phones. To overcome the difference, the *context dependency transducer* is introduced. Assuming the triphone model, this WFST can be constructed by creating a state for every pair of phones, with transitions labeled by newly recognized input phones. Figure 2.6 shows an example

[3]words that are pronounced the same way, such as 'read' and 'red'

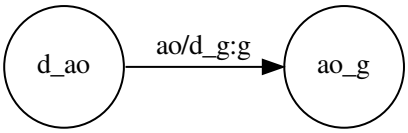


Figure 2.6: A single transition in a context dependency transducer. It depicts the situation, where the triphone ao/d_g ('ao' preceded by 'd' and followed by 'g') is recognized and a context-independent phone 'g' is emitted.

transition. This WFST is usually referred to as C and has no information encoded in the weights of transitions.

When we compose all these three components together, the resulting transducer $C \circ L \circ G$ maps context dependent phonemes to word sequences constrained by G.

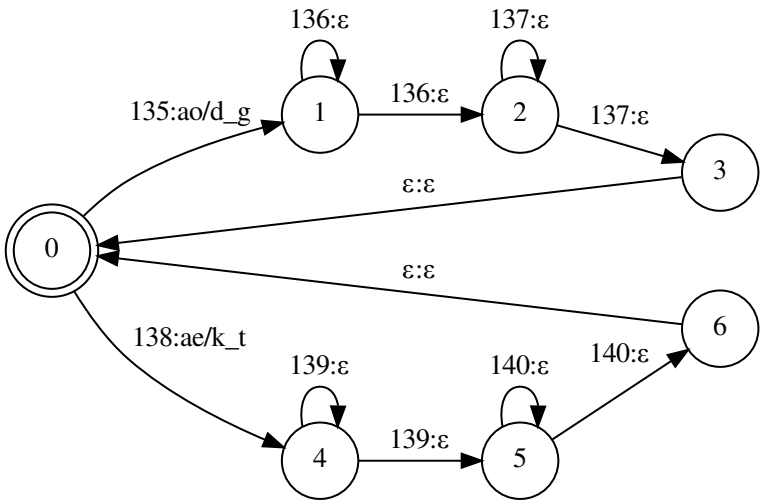


Figure 2.7: A example fragment of H transducer, with weights omitted for simplicity. Triphones 'ao/d_g' and 'ae/k_t' are shown. The integer input labels refer to emitting probability density functions.

The HMM topology is a transducer corresponding to the HMM state sequences for single context dependent phones. It translates output probability density functions to context dependent phones. For phones, a three state structure is usually adopted (for instance in the Kaldi toolkit [18]), as is illustrated in figure 2.7. A more complex model, involving inner loop, is usually used for silence, as it can contain different non-speech sounds.

We can get the complete recognition graph as HCLG $= H \circ C \circ L \circ G$. For practical

reasons, we want this transducer to be deterministic and minimal, so the usual recipe for decoding graph construction is: $\delta(min(det(H \circ det(C \circ det(L \circ G)))))$ [15], where *min* and *det* represent weighted minimization and determinization respectively. The operation δ represents the final removal of disambiguation symbols, which is done by replacing them by epsilons.

2.4 Decoding in such a network

As already stated in section 2.3, the HCLG recognition network can be considered to model the structure of a big HMM. Therefore, the decoding is typically (e.g. in the Kaldi toolkit) implemented as the token passing algorithm described in section 2.1.

For large vocabulary continuous speech recognition (LVCSR), the recognition network can get very large (several millions of arcs), mainly because of the language model G. Then various techniques of pruning are incorporated into the decoder in order to keep the process of decoding reasonably fast. For example, tokens can be kept for further processing, only if they have score close enough to the current best token. The idea is, that those, that are much worse than the best, would not produce successful tokens in the next step, so there would be no point in deriving lots of tokens from them, only to discard them right after.

For the batch processing of speech, speed is a sufficient property to optimize. However, when recognizing on-line, real-time latency is of interest as well. Therefore a technique has been implemented in the Kaldi toolkit to retrieve the results of recognition as soon as possible, instead of processing the whole file at once. The concept of the *immortal token* is introduced. It is such a token, that all active tokens have been derived from it. Since every token has a single parent, it is quite simple to check by backtracking, whether there is some common ancestor of active tokens. When a new immortal token is found, recognized sequence is retrieved by backtracking from the current immortal token to the previous one.

This way, the latency could be theoretically reduced to processing a single feature vector, corresponding typically to 10 milliseconds, but in practice, the online decoder works with batches in order of lower tens of feature vectors.

2.5 Recognition performance metric

In automatic speech recognition (ASR), word sequences are the target. Therefore a certain kind of string distance is used. The most common metric is Word Error Rate (WER). According to [11], the WER is defined as follows:

$$\text{WER} = \frac{S + D + I}{\text{length of the correct transcription}}$$

where S is the number of substitutions, D is the number of deletions and I is the number of insertions. The WER is typically given in percents, sometimes *word accuracy* $= 1 - \text{WER}$ is used. Sometimes the word accuracy is defined not to consider insertions. The correct evaluation of this metric requires solving maximum substring matching problem, which is straight-forwardly solved by means of dynamic programming [11].

Another metric used is the *sentence error rate* (SER). The SER is defined simply as

$$\text{SER} = \frac{\text{E}}{\#\text{sentences}}$$

where E is the number of sequences, which were recognized incorrectly. We shall not discuss what a *sentence* could mean in recognition of longer utterances. In this work, short phrases will be recognized, where we can claim a sentence to be equal to an utterance.

Chapter 3

Turning context free grammars into WFST specified language models

As shown in section 2.4, the finite state representation of a language model allows for efficient decoding. On the other hand, it is often more convenient to describe grammars in a more flexible manner. We can consider the case when a grammar of a command and control system requires the user to specify a time interval by giving two dates. Expressing this grammar in a strictly finite state manner requires having the subautomaton specifying the date twice, which introduces undesirable redundancy. In this simple case, the redundancy could be avoided by introducing some simple substitution mechanism.

Yet for general cases, possibly including cycles or indirect recursion, human inspection is inefficient and error-prone. Therefore, a tool has been developed to do this task automatically. This chapter describes the design of this tool, of arbitrary name *Grammator*. Instead of developing a new grammar specification format, the Speech Recognition Grammar Specification [12] was used as the input format. The AT&T FSM format [3] was chosen as the output, for it is the standard textual format[1] of WFSTs used in Kaldi.

3.1 Speech Recognition Grammar Specification

The W3C recommendation [12] defines the Speech Recognition Grammar (SRG) as a syntax for grammar representation. Two specific forms are accepted: Augmented BNF syntax (ABNF) and XML. For the current version of *Grammator*, only XML was taken as supported format, however, these formats are semantically mappable. Further discussion of the SRG will be with respect to its XML format. SRGs are defined with the expressive power of context free grammars (CFG), so most of the effort here went into reducing it to the power of (weighted) finite state transducers (WFSTs).

Within the SRGS, a *grammar* (always the root element) is given as a set of *rules*. There is no theoretical limit on the number of rules, and they can be defined with either global or document-only scope. A rule is an analogy of a nonterminal symbol in the formal CFG and the specifies its *root rule* (likewise to the starting nonterminal of a formal grammar). Content of the `<rule>` element is its expansion, which is in general a string of *items*. Items can be explicitly specified using the `<item>` element or implicitly as sequence of words.

[1] for most of the WFST operations, Kaldi uses the OpenFST toolkit [4], which in turn uses the AT&T format.

When defined explicitly, repetition can be specified, of either given, or possibly infinite number of times. Furthermore a `<one-of>` element can be used to list several alternatives.

Finally the `<ruleref>` element is used to reference to another rule, meaning the expansion of the referenced rule shall be inserted into the spot where it occurs. Except for user-defined rules, special ones are already predefined. These special rules match either any utterance, nothing, or a zero-length utterance. This will be further discussed in section 3.4.

There are two different flavours of weights in the SRGS. At first, alternatives in the `<one-of>` element can be assigned weights, which have no exact interpretation specified, only 1 is specified as the neutral weight, with greater weight making the alternative more likely. Also any repetition can have a "repeat" probability specified. These are to be interpreted as the probability, that the item will be repeated yet another time, independently of the number of repetitions so far, except for the total number of repetitions has to stay in the given bounds.

There are *tokens* defined in SRGS, which are supposed to carry some semantic interpretation, however there is neither a natural expression of these in WFST, nor use for such feature in our current system, so tokens are ignored when forming an equivalent grammar.

As will be shown in chapter 4, not the full power of a CFG is necessary for all applications. So a design decision was made, that only those grammars will be supported, that do not exceed the power of regular languages. Another possibility was to approximate a CFG by either its subset or superset as discussed in [17], but there was no use case for it in this project, so it has been left as an option for further work.

3.2 Transformation into Context Free Grammar

Even though the SRGS has the expressive power of a CFG, it introduces several convenience features, that make the process of loading it as a CFG not straight forward. Therefore, this process will be described briefly. At this level, each rule can be turned into a nonterminal independently of others, which is not the case with the following transformation into WFST.

At first, we consider rules with no repetitions, i.e. such rules, that can only expand into a finite number of strings (over the total alphabet). We could either list all these strings and then simply turn them into right-hand sides of rules in the resulting grammar, which would keep the number of nonterminals the same, but at the cost of increasing number of expansions. The number of right-hand sides per nonterminal would then follow $O(N^K)$, where N is (an average) number of options per `<one-of>` element in the rule and K is number of these elements in the rule expansion. The other option is to introduce a synthetic nonterminal for each `<one-of>` element, encapsulating all the options in it. This way, the number of nonterminals is increased by $O(K)$, but the total number of rules in the grammar does not exceed $O(NK)$. The structure of the original SRGS is also better reflected in the grammar this way, so this method was implemented in the Grammator. The difference is illustrated on a simple example in table 3.1 (page 14).

In case a limited number of repetitions is specified, we define a rule for each number of repetitions. If the number of repetitions is unbounded, first, a chain of repetitions corresponding to the lower bound is put at the begining of the rule. Then, a new nonterminal is introduced which handles the possibly infinite recursion. An example is shown in the table 3.2 (page 14).

Considering the special rules, each is handled separately. The "garbage" rule, matching any utterance, is not supported, because no use has been found for it and, more importantly, it would inevitably introduce nondeterminism in decoding, since there would be no way to

Table 3.1: Demonstration of two possible transformations of a SRGS rule with alternatives into equivalent formal CFG rules. In the Grammator, the one in the third column is applied for its better space complexity and structure, that follows the original SRGS more closely.

`<rule id="A">`	A → aKx	A → A-1 A-2 A-3
`<one-of>`	A → aKy	A-1 → a
`<item>a</item>`	A → bKx	A-1 → b
`<item>b</item>`	A → bKy	A-2 → K
`</one-of>`	A → aLx	A-2 → L
`<one-of>`	A → aLy	A-3 → x
`<ruleref uri="#K"/>`	A → bLx	A-3 → y
`<ruleref uri="#L"/>`	A → bLy	
`</one-of>`		
`<one-of>`		
`<item>x</item>`		
`<item>y</item>`		
`</one-of>`		
`</rule>`		

Table 3.2: Transformation of an item with unbounded repetition into a CFG.

`<rule id="A">`	A → xxA'
`<item repeat="2-">`	A' → x
`x`	A' → ϵ
`</item>`	
`</rule>`	

tell before processing whole utterance, whether the recognized word is a grammar sequence or should still be captured as garbage. For the on-line decoding (see the use case in chapter 4), this would be even worse, because we want to display recognized words as soon as possible. The "void" rule, which does not match anything, is expressed as a nonterminal with a single expansion rule $A \rightarrow A$, that is an infinite recursion, which clearly matches nothing. Finally, the "null" rule, which stands for an empty utterance, is simply skipped when constructing the grammar.

Both the weights of alternatives and repeat probabilities are reflected in the grammar. To achieve this, each production rule of the grammar has a weight associated. In order to keep the transducer stochastic and avoid mixing weights and probabilities, weights are normalized to sum up to one for each nonterminal.

3.3 Deciding on the power of the grammar

Since only a subset of CFGs can be equivalently expressed as a WFST, the given grammar must first be tested to determine, if it can be transformed. To decide whether a given grammar can be expressed in terms of WFST, we first consider the close-to-exact answer. Recall that a CFG is defined as a 4-tuple:

$$(N, T, P, S)$$

where:

N	is a finite set of nonterminal symbols
T	is a finite set of terminal symbols, $T \cap N = \emptyset$
$P \subseteq N \times (N \cup T)^*$	is the set of rules of the form $A \to u$
$S \in N$	is the initial nonterminal

We also define the relation of one *derivation step* $uAv \Rightarrow uxv$, for strings u, v, x over the total alphabet $N \cup T$ and any nonterminal A such, that $A \to x \in P$. Then its transitive-reflexive closure $x \Rightarrow^* y$ represents that the string y can be *derived* from x in an arbitrary number of steps.

Definition 1. *A context free grammar G is* self-embedding *when there is a nonterminal A such, that $A \Rightarrow^* uAv$ for some non-empty strings u, v.*

It has been proven (see [6]), that if a CFG is not self-embedding, an equivalent finite automaton can be found. It has also been shown, that some of the self-embedding CFGs do not exceed the expressive power of finite automata (see [5]). However, these are not taken into account so far, as they would greatly broaden the scope of this work without an actual need for it.

An even stronger restriction is put on the CFG to be processed by the tool. Grammator allows no nonterminal symbol A, such that $A \Rightarrow^* uAv$, for a non-empty string v. This effectively means banning left recursion, which is a natural condition in formal language processing, because a (common and efficient) top-down parser cannot process it [14]. A similar decision has been made in Microsoft's tool Whisper [11] and an example in chapter 4 demonstrates, that this does not decrease the practical value of Grammator.

To test whether a given CFG is neither self-embedding nor left-recursive, I have introduced a relation $Q \subseteq N \times N$:

$$Q = \{(A, B) \mid A \to uBv \in P,\ u, v \in (N \cup T)^*,\ v \neq \epsilon\}$$

This relation captures which nonterminals B may appear at any-but-last position in a sentence. Armed with this knowledge, we ask whether this B can ever turn into an A, which would violate the requirements. To express this possibility, I have introduced another relation $D^+ \subseteq N \times N$:

$$D^+ = \{(A, B) \mid A \Rightarrow^+ uBv,\ u, v \in (N \cup T)^*\}$$

Constructing this set directly is generally impossible, because it is defined using the possibly infinite relation $\Rightarrow^+$. So we first define relation $D = \{(A, B) \mid A \to uBv \in P\}$ and then take D^+ as its transitive closure.

Then the composition $D^+ \circ Q$ represents, for a given $(A, B) \in D^+ \circ Q$, that a such string can be derived from A in an arbitrary number of steps, that B occurs in it, but not at the last position. Therefore, the requirement of a grammar being neither self-embedding nor left recursive is equivalent to this relation being anti-reflexive.

Definition 2. *A context free grammar G is* Grammator-compliant *when the relation $D^+ \circ Q$ is anti-reflexive.*

3.4 Construction of Weighted Finite State Transducers from Weighted Context Free Grammars

After a grammar is checked to meet the requirements, a WFST is constructed from it. At first, an intuition to this process will be given on a simple example. Consider a grammar

S with $N = \{E, O, F\}$, $T = \{a, b, +, -\}$, starting nonterminal E and rules: $E \to OFO$, $O \to a$, $O \to b$, $F \to -$ and $F \to +$. Although it is a CFG from the formal point of view, it does not exploit the expressive power of a CFG – an equivalent finite state automaton is shown in figure 3.1. However, the construction of such automaton is considerably less straight forward than in the case of constructing it from a right- or left-linear grammar.

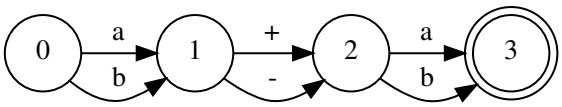


Figure 3.1: A simple FST corresponding to `{a,b}{+,-}{a,b}`

With a right-linear grammar[2], fragments of the WFST can be constructed for each nonterminal independently and then interconnected according to the nonterminal at the end of the right-hand side of each rule. This is not the case with (Grammator-compliant) CFGs. Considering the nonterminal O from grammar S, we can see, that it is instantiated in two distinct parts of the corresponding FST – the first in transition $0 \to 1$, the second in transition $2 \to 3$. To properly construct each of these fragments, more information is needed than only the sequence of symbols in the right-hand side of the rule.

Putting aside issues of recursion for a while, we can construct a fragment of WFST corresponding to a given nonterminal easily, but it has to be plugged into the parent WFST. To achieve this, each of these fragments has a well defined entry- and exit-point. This way, the WFST can be constructed "on-the-fly" recursively, beginning with the starting nonterminal and constructing the WFST fragments as needed. Then, the "parent" fragment construction is responsible for correctly connecting to these endpoints.

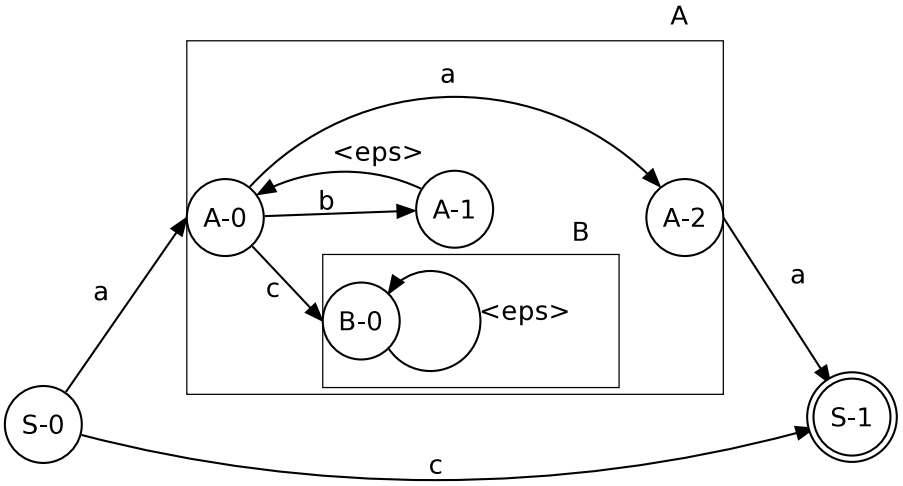
Recursion in the grammar has to be handled with care. Firstly, consider that we keep a global stack of nonterminals, which are currently being synthesised and already have a defined entry point. We add their symbol to the stack when beginning their construction and remove it when the construction of the particular WFST fragment is done. Upon finding a nonterminal A at the end of the right-hand side r of the currently processed rule, we could consult this stack. If A was there, we would handle the recursion by simply connecting the end of r to the entry point of A (otherwise we just construct a new WFST fragment corresponding to A).

However, this method allows invalid result for cases, when r is not the rightmost part of the derivation of A. E.g. consider a grammar, where there are rules of type $A \to uBv$, $A \to x$ and $B \to A$. Then the method explained above would connect the beginning of B to the already constructed entry point of the A. Even though this is clearly not correct, because it allows accepting a (sub)string ux, it is not an actual problem, because a Grammator-compliant grammar can not exhibit such behaviour – this is an example of self-embedding.

[2]Left linear will not be discussed further as they introduce left recursion, however their behaviour is similar.

$S \rightarrow aAa$
$S \rightarrow c$
$A \rightarrow bA$
$A \rightarrow a$
$A \rightarrow cB$
$B \rightarrow B$

(a)



(b)

Figure 3.2: An example of WFST 3.2b constructed from a grammar 3.2a. This example does not precisely follow algorithm 1, some ϵ-transtitions are omitted for clarity. The rule $S \rightarrow aAa$ is implemented by the upper part of the WFST, the big rectangle represents a WFST-fragment corresponding to A. For the construction of the WFST fragment corresponding to S, A-0 and A-2 are the entry- and exit-points respectively. Within it, the recursive part of the rule $A \rightarrow bA$ is implemented by the transitions A-1 $\rightarrow$ A-0, because A-0 is now available as the entry-point for the currently being synthesized nonterminal A. The fragment of WFST corresponding to the nonterminal B demonstrates the situation, where the special rule "void" was given in the specification and thus no utterances shall be recognized with it.

However, a slight change to the algorithm has been introduced anyway. For implementation purity, the stack structure of nonterminals being currently synthesized is not global. When a nonterminal is derived as the rightmost symbol, the procedure of constructing the corresponding WFST fragment is given the full stack of nonterminals currently synthesised as before. In the other case, the nonterminal-handling procedure could not use those nonterminals anyway, so an empty set is passed.

More formally, this process can be described by a function (algorithm 1, page 18) that constructs a WFST fragment from a single nonterminal. A non-formal commented example is given in the figure 3.2.

In order to start the construction of the whole grammar, an empty stack of nonterminals usable for recursion and an empty state are passed as arguments to the function constructing the starting nonterminal of the grammar. This pre-constructed state then becomes the initial state of the WFST. The only final state is the exit point of the starting nonterminal.

3.5 Implementation and output format

The Grammator tool implements the above described principles in a straight forward way. The computation is organized in a simple object-oriented schema, with a separate class responsible for each of the models involved.

At first, there is the class `XMLTree`, which keeps the XML tree representation of the SRGS. To read it from an XML file, it uses the standard `ElementTree` module. This class

Algorithm 1 Synthesize a WFST fragment corresponding to the given nonterminal

Require:

- A is a nonterminal of a Grammator-compliant input grammar G
- *nonterm_ stack* is a set of currently being synthesized WFST fragments
- *predecessor* is the entry point of the resulting fragment
- the function connect(*src*, *dst*, *lab*, w) creates an arc from state *src* to state *dst* with label *lab* and weight w

Ensure:

- an equivalent WFST fragment is constructed for the nonterminal A
- its entry point is the given *predeccesor* state
- the reference to the exit point of the fragment is returned

```
1: function MAKEFRAGMENTWFST(A, nonterm_stack, predecessor)
2:     exit_point ← newState()
3:     for all s ∈ rhs(A) do
4:         last ← newState()
5:         connect(predecessor, last, ϵ, weight(s))        // here weights get in
6:         for all elem ∈ s do
7:             if nonterminal(elem) then
8:                 if elem ∈ nonterm_stack then        // recursion
9:                         // connect to the entry point of elem on stack
10:                    connect(last, entryState(find(elem, nonterm_stack)), ϵ, 1̄)
11:                    last ← nullState()
12:                else      // not a recursion, construct new fragment
13:                    if lastElement(elem, s) then
14:                        pn ← nonterm_stack ∪ {A}
15:                    else
16:                        pn ← ∅
17:                    end if
18:                    last ← makeFragmentWFST(elem, pn, last)
19:                end if
20:            else      // terminal symbol
21:                ns ← newState()
22:                connect(last, ns, elem, 1̄)
23:                last ← ns
24:            end if
25:        end for
26:        if last ≠ nullState() then
27:            connect(last, exit_point, ϵ, 1̄)
28:        end if
29:    end for
30:    return exit_point
31: end function
```

is responsible for transforming the tree into an equivalent CFG, which is represented by the class `Grammar`. This class is mainly responsible for providing various tests, such as whether it is complete (no symbols referenced in the rules are missing), whether it is right-linear and finally whether it is Grammator-compliant. The final representation in Grammator is the WFST, which is captured by the class `GeneralFST`. It knows how to construct itself from a `Grammar` object and in turn to output itself to the AT&T syntax.

The AT&T syntax of finite state machines is quite simple. Each state is identified by a non-negative integer. Terminal symbols are identified by non-negative integers as well, but the number 0 is reserved for the empty symbol (ϵ). For every arc in the WFSTs, there is a line of the form:

S D I O C

where S refers to the source state, D to the destination state, I is the number (integer) of the input symbol of the arc and O is the number (integer) of the output symbol. C is a floating point number representing the arc cost. The source state of the first arc in the FSM description is, by convention [3], the initial state of the WFST. Finally, each final state is specified by a line

S C

where S is the number of the final state and C is the cost of accepting the string in this state. The mapping of symbols (typically words for language modeling) to numbers is kept in a separate file, referred to as *symbol table*.

Since the task of the Grammator is inherently off-line (related to the decoding) and therefore has loose requirements on speed, it was implemented in the Python scripting language.

Chapter 4

Use Case: The A-PiMod project

The techniques described in chapter 3 were applied in the A-PiMod project. The A-PiMod project (Applying Pilot Model for Safer Aircraft) [2] is a European Committee-funded project aimed at increasing the safety of flight by observing the crew state and properly responding to it, by means of task distribution between the crew and advanced automation systems.

One of the target modalities of Human-Machine interaction in this project is speech. For the first part of the project, the partial goal is to have a recognizer of grammar constrained phrases for voice control of the cockpit. Other partner in the project, Honeywell CZ, is responsible for the grammar itself, I have developed a methodology for using such grammar as the language model in speech recognition.

The application of the developed techniques in the A-PiMod project requires a lot of effort in integration, but only the speech recognition part itself is described in this thesis for brevity.

For completeness, the structure of the used acoustic models is briefly described in section 4.3. The applied techniques are not described in this thesis, so the reader is encouraged to study them in relevant the literature. The Gaussian Mixture Model is a generative statistical model well described in [7]. The Linear Discriminant Analysis can be interpreted in several ways, but it is usually understood as a linear projection of the data into a subspace of given dimension, where the classes are best separated. It is also described in [7]. The Cepstral Mean and Variance Normalization is a simple method for increasing the robustness of the features, first presented in [19].

4.1 Input grammar

Due to the nature of the project, there are several separate grammars involved. The grammar to be used for recognition is to be given by the *interaction context* (IC). Even though the on-line change of the IC is not implemented so far, support is prepared for them. The challenge is, that the voice control in each IC is given by two parts: Global rules, which hold valid for any IC, but refer to different objects according to the IC (an example is given in table 4.1) and local rules, which refer directly to objects in the particular IC (an example is given in table 4.2).

The root element of the grammar is named `<input>` and refers to one or more of the global rules, as well as several local rules, thus it is a part of the local definition of the grammar. Even though the Speech Recognition Grammar Specification (SRGS) allows

cross-file reference, for global rules, that refer different objects according to the IC, some kind of back-reference would be needed, where the rule would be aware of which grammar (IC) invoked it, so that the right objects could be picked in the global rules.

Table 4.1: Example of the *global show* rule in the A-PiMod project. There are also *global get*, *global set* and *global hide* rules of similar structure. The rules *showable object*, *showable object with parameter required* and *parameter* are given by the interaction context.

```
<rule id="global_show">
 SHOW
 <one-of>
  <item><ruleref uri="showable_object"/></item>
  <item><ruleref uri="showable_object_with_parameter_required"/>
   <ruleref uri="parameter"/></item>
 </one-of>
</rule>
```

Such behaviour is not defined by SRGS, so in order to avoid redundancy, the following approach has been taken: Global rules are stored in a separate XML-fragment, which contains the header and global rules definition, while the fragment corresponding to the IC does not include the XML header, but is properly ended with the `</grammar>` tag. Before processing them by Grammator, these two parts are concatenated into a valid XML file describing possible phrases in the given IC.

Another possibility would be broaden the SRGS in such a way, that some part of rule reference would become parametric. However, this would prevent grammars in such format from being processed by some other tool expecting the SRGS format.

Table 4.2: Example of local rules in the A-PiMod grammar. It is taken from the Interaction Context "Cross Dialog". The rule `number` defines possible altitudes by enumeration.

```
<rule id="set_altitude">
 SET ALTITUDE <ruleref uri="#altitude_settings"/>
</rule>
<rule id="altitude_settings">
 <one-of>
  <item>ON</item>
  <item>OFF</item>
  <item>AT OR BELOW<ruleref uri="#number"/></item>
  <item>AT<ruleref uri="#number"/></item>
  <item>AT OR ABOVE<ruleref uri="#number"/></item>
 </one-of>
</rule>
```

Once the weighted finite state transducer equivalent to the given grammar has been constructed, it can be used as a language model like any other. This way, the obtained language model can be relatively easily combined with custom acoustic models to create the desired system, and then, using the same acoustic model, compared to statistical language models.

4.2 Overview of relevant parts of the Kaldi toolkit

Kaldi [18] is a speech recognition toolkit aimed at researchers. Its core consists of object-oriented code in C++, which implements most of the state-of-the-art techniques used in speech recognition. As the basic access to this technology, simple command-line applications are developed, that conduct single operations, such as "perform this linear transform on these features" or "given these acoustic scores and this recognition network, decode". These tools have been developed with pipelines in mind, so using a shell script to build a system is a natural choice. However, it is possible to avoid using these applications and compile a bigger one. This is the case with the A-PiMod project, where, unlike in research, real-time processing is important, so the whole recognizer is compiled into one application.

The Kaldi project not only provides recognizer technology, but comes also with utilities and recipes for building systems on widely available corpora. The Kaldi recipe for constructing recognition networks is among these. It is a bit different from the classical one described in section 2.3: The final recognition network is (following the notation of chapter 2) constructed as:

$$\mathrm{HCLG} = \alpha(min(\delta(H' \circ min(det(C \circ min(det(L \circ G)))))))$$

where the transducer H' is similar to the HMM topology introduced in section 2.3, but without the self-loops. The final α operation adds the self-loops. Without the self-loops, the composition of the CLG transducer with the HMM topology puts lesser requirements on the computing system, especially the memory.

There is also an implementation of an on-line recognizer in Kaldi. Its structure is captured in figure 4.1. The online audio source can be implemented in different ways to support various inputs, such as from pre-recorded wavfiles, an input audio device or a TCP socket. It outputs raw audio samples. These are then processed by a pipeline of feature processing blocks. In the current systems, the first block segments the audio into frames and computes either Mel-Frequency Cepstral Coefficients (MFCC) [9] of Perceptual Linear Prediction (PLP) [10] coefficients. Furthermore, implementation is provided for on-line cepstral mean normalization, computation of delta and delta-delta features and application of linear transformation.

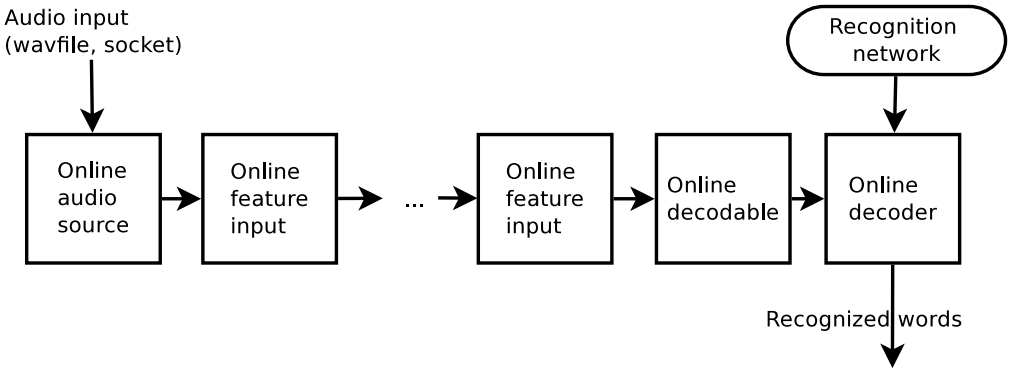


Figure 4.1: Block schematic of the on-line recognizer in Kaldi. The blocks named "online feature input" perform various feature transformations, such as adding the delta coefficients or applying linear transformations.

On top of this feature-processing pipeline is "online decodable". *Decodable* is an important abstraction in Kaldi. An object of this type provides acoustic scores to the decoder.

More precisely, it produces the log-likelihood for a given frame under a given distribution. This way, it encapsulates the interaction of the acoustic model with the features, allowing the decoder to work completely unaware of the type of the acoustic model. When the decoding is online, the decodable can not provide log-likelihoods for any frame of the speech recording, but only for the current frame and for a short history before it – in the default setup for last 27 frames.

4.3 Acoustic models used for recognition

The voice part of the human-machine interface in the A-PiMod project is designed to accept English speech. However, the pilots are generally not native English speakers, so English of various accents needs to be properly recognized. Therefore, it was reasonable to search for an acoustic model trained on non-native English. One such model was trained at Speech@FIT by Kateřina Žmolíková on data obtained during the AMI/AMIDA project. The data corpus is distributed with the project [1].

With respect to the speech recognizer scheme (2.1), the audio input is processed in the following way: At first, the speech is segmented into frames and PLP coefficients are computed for each of these. The cepstral mean and variance normalization (CMVN) is then applied on these feature vectors. Further on, nine feature vectors are spliced together and a linear transformation – the Linear Discriminant Analysis – is applied. The features obtained this way form the sample space of the Gaussian Mixture Model (GMM), which is applied as the acoustic model. For computational reasons, their covariance matrix is limited to a diagonal one. The block schematic of this computation is given in figure 4.2.

Figure 4.2: Block schematic of feature extraction used in the AMIDA-system.

Since the AMIDA-system was available at Speech@FIT, I could simply change the grammar model (WFST G) to the one corresponding to the desired Interaction Context and recompile the network HCLG. Therefore, the network is fully compatible with the corresponding acoustic model and can be directly used for recognition.

4.4 Baseline language model

Since only the language model was changed, I could compare my system to other ones with the same acoustic model. However, a reasonable language model had to be chosen. As this thesis presents a grammar-based language model in contrast to n-gram language models, the n-gram language model was a natural choice. Because the AMIDA project was aimed at automatic speech recognition of meetings, it should be suitable for short utterances. It also covers a very wide vocabulary, so it can be expected to understand control words as well.

To improve this baseline model, I have taken a model created by Kateřina Žmolíková at Speech@FIT. It is the basic AMIDA language model interpolated with a bigram model trained on approx. 1300 utterances of average length of ten words, capturing cockpit-ATC[1]

[1] Air Traffic Control (ATC) is the common name for the service provided by the ground personnel of an airport to prevent collision in the air. The ATC Tower (ATCT) is the communication point for this service.

communication. Later in this chapter, this model is referred to as the *airspeak* model.

The resulting recognition network is very large: the size of the file containing the binary representation of the HCLG is approx. 1 GB. Because OpenFST does not support memory-mapped files, this imposes quite high memory requirements. Also, loading the model can take several seconds. However, should this model prove successful, it would be sufficient for any Interaction Context, so no need would arise to switch the recognition network when changing the IC.

4.5 Experiments and results

For the experiments the Main Screen was used, which is the currently largest Interaction Context defined. Thirty-three random phrases were generated from it, using the OpenFST tool `fstrandgen`. Then three speakers read them aloud. One of them was a male Czech, one a male German and the last one a female Russian. This way we can expect the test set not to be affected by a particular accent.

These utterances were recorded in a silent environment. Since pilots are used to push-to-talk devices, I have trimmed the utterances manually, so that approximately half a second of silence precedes and follows the speech. This has been claimed to be a reasonable robustness requirement by the Honeywell partners. The advantage of this approach is, that no Voice Activity Detection needs to be run on the data.

As the basic metric, word error rate has been evaluated, as is captured in table 4.3. The

Table 4.3: Word error rate using different systems. Each column represents recordings from one speaker, the speaker is labeled as <nationality>_<gender>. The first row (airspeak) is the baseline system using the n-gram model.

System	CZ_M	DE_M	RU_F
airspeak	83 %	68 %	65 %
grammar	14 %	9.0 %	5.0 %

sentence error rate has been evaluated as well, results are in the table 4.4. The SER can be considered the more important metric for the use-case, because the recognized sentence is to be used as the unit for further processing.

To understand how it is possible, that for the grammar based system the WER is consistently higher than the SER, we must take into account, that a single miss on the sentence level typically means, that most of the words were recognized incorrectly. And since simpler and shorter sentences were recognized better, this results in a higher ratio of words mis-recognized.

Table 4.4: Sentence error rate using different systems. Each column represents recordings from one speaker, the speaker is labeled as <nationality>_<gender>. The first row (airspeak) is the baseline system using n-gram model.

System	CZ_M	DE_M	RU_F
airspeak	73 %	79 %	82 %
grammar	12 %	6.1 %	3.0 %

We can see, that the recognizer using the IC grammar outperforms the one based on trigram language model by far. This is no surprise, as the grammar represents the actual distribution, from which the sentences were drawn, while the trigram model is only a very general approximation. To check, whether the system based on the airspeak language model does work at all, we can have a look at some of its erroneous outputs. Three examples are given in table 4.5. We can consider the recognized output to be quite similar to the true transcription in terms of acoustics, so the system seems to operate properly to some extent.

Another reason, why this language model performs so poorly, is quite likely the structure of the phrases. The training corpus for AMIDA contains spontaneous speech, which was often grammatically incorrect and was often interrupted during the meetings, yet it was very much following the usual structure of English sentences. On the other hand, the phrases in A-PiMod ICs do not depend on any context, which could be exploited by the recognizer, and follow a simple imperative schema "<do> <something>".

Table 4.5: Example of the kind of errors made by the airspeak-AMIDA system on utterances from the Czech male speaker. It can be seen, that the recognized phrases are somewhat acoustically close to the true ones.

True transcription	airspeak-amida system output
SHOW CONTEXT MENU	SHOW QUOTED MEMO
FULL SCREEN	WHO SCREAM OH
HIDE CROSS	WHY TO CRUISE

Chapter 5

Conclusion

In this Bachelor Project, I have developed a tool called *Grammator* used for the transformation of a substantial subset of Context Free Grammars into Weighted Finite State Transducers, for purposes of language modeling in speech recognition. This thesis describes the design of this tool, as well as the theoretical background of the use of Weighted Finite State Transducers in speech recognition.

To test the language model obtained this way, an example task from the A-PiMod project was used. Using the Grammator, I have built a language model for a particular cockpit situation. Then, I combined this model together with an acoustic model from a system trained on non-native English. Finally, this system was compared with a system using a general purpose English trigram language model. The results were greatly in favor of the language model derived from the grammar. However, these experiments are rather a preliminary proof of concept than a full result.

The outcome of this project will be further used in the A-PiMod project. The Grammator tool is used as one of the core elements in the creation of a system for a given cockpit situation. Should the need arise, the tool can be enhanced by methods for constructing WFSTs that approximate even such CFGs, which can not be transformed directly, because their expressive power exceed the finite state automata.

Further effort will be invested in the recognition of grammar fragments, so that other modalities can provide some input into the decoder. The ultimate goal in this direction is to allow arbitrary interleaving of modalities in the input to the system, e.g. the pilot could say "set" command, then pick some object via a touchscreen and then say a requested value of some parameter.

The long term vision includes an online decoder operating directly on the grammar, which would allow arbitrary CFG to be used as the language model. Also a solid detection of out-of-grammar utterances would be a useful extension to the recognition system.

Bibliography

[1] AMI meeting corpus. https://www.idiap.ch/dataset/ami, 2010.

[2] A-PiMod: Applying pilot models for safer aircraft. http://www.apimod.eu/Default.aspx, 2013.

[3] Cyril Allauzen, Mehryar Mohri, Fernando Pereira, and Michael Riley. The AT&T FSM format. http://www2.research.att.com/~fsmtools/fsm/man4/fsm.5.html. Version 4.0.

[4] Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. OpenFst: A general and efficient weighted finite-state transducer library. In *Proceedings of the Ninth International Conference on Implementation and Application of Automata, (CIAA 2007)*, volume 4783 of *Lecture Notes in Computer Science*, pages 11–23. Springer, 2007. http://www.openfst.org.

[5] Stefan Andrei, Wei-Ngan Chin, and Salvador Valerio Cavadini. Self-embedded context-free grammars with regular counterparts. *Acta Informatica*, 40(5):349–365, 2004.

[6] Noam Chomsky. On certain formal properties of grammars. *Information and Control*, 2(2):137–167, June 1959.

[7] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0-387-31073-8.

[8] Cyril Allauzen and Mehryar Mohri. Efficient algorithms for testing the twins property. *Journal of Automata, Languages and Combinatorics*, 8(2):117–144, 2003.

[9] Steven B. Davis and Paul Mermelstein. Readings in speech recognition. pages 65–74. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.

[10] Hynek Heřmanský. Perceptual linear predictive (PLP) analysis of speech. *J. Acoust. Soc. Am.*, 57(4):1738–52, April 1990.

[11] Xuedong Huang, Alex Acero, and Hsiao-Wuen Hon. *Spoken Language Processing: A Guide to Theory, Algorithm, and System Development*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2001.

[12] Andrew Hunt and Scott McGlashan. Speech recognition grammar specification version 1.0. W3C recommendation, W3C, March 2004. http://www.w3.org/TR/2004/REC-speech-grammar-20040316/.

[13] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, volume 77, 1989.

[14] A. Meduna. *Elements of Compiler Design*. Computer science. Computer engineering. Computing. Taylor & Francis, 2007.

[15] Mehryar Mohri and Fernando Pereira and Michael Riley. Speech recognition with weighted finite-state transducers. In *Springer Handbook on Speech Processing and Speech Communication*, chapter 28. Springer-Verlag New York, Inc., 2008.

[16] Tomáš Mikolov. *Statistical Language Models Based on Neural Networks*. PhD thesis, Brno University of Technology, 2012.

[17] Mark-Jan Nederhof. Practical experiments with regular approximation of context-free languages. *Computational Linguistics*, 26(1):17–44, March 2000.

[18] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, Jan Silovsky, Georg Stemmer, and Karel Vesely. The kaldi speech recognition toolkit. In *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*. IEEE Signal Processing Society, December 2011. IEEE Catalog No.: CFP11SRW-USB.

[19] Olli Viikki and Kari Laurila. Cepstral domain segmental feature vector normalization for noise robust speech recognition. *Speech Communication*, 25(1-3):133–147, August 1998.