TU Kaiserslautern & DFKI
Image Understanding and Pattern Recognition
Prof. Dr. Thomas Breuel

Projektarbeit

# Bibliographic Meta-Data Extraction Using Probabilistic Finite State Transducers

*Martin Krämer*

Matrikelnummer 346493

15.12.06

Betreuer: Hagen Kaprykowsky

# Contents

## Abstract

*In this paper we present the application of probabilisitc finite state transducers to the problem task of bibliographic meta-data extraction from research paper references. Although finite state techniques have been utilized on various tasks of computational linguistics before they have not been used for the recognition of bibliographic references yet. Especially the involved simplicity and flexibility of modelling as well as the easy adaptability to changing requirements turn out to be beneficial. An evaluation on the Cora dataset that serves as a common benchmark for accuracy measurements and represents quite "hard" cases yields a word accuracy of 88.5%, a field accuracy of 82.6% and an instance accuracy of 42.7%. Therefore our system performs second best on the given testset regarding the published results of similar projects.*

## 1 Motivation

Research paper search engines like CiteSeer [GBL98, LGB99b, LGB99a, GMLG01] become increasingly important nowadays as they enhance researchers' efficiency due to a faster access to current resources and the possibility for quicker distribution of new releases. Furthermore hiring decisions become more and more influenced by the research papers found via such a search engine given the applicant's name [PM04].
And as the quality of results is directly dependent on the quality of the information extraction component – which in our case tries to recover bibliographic meta-data[1] (i.e. BibTeX subfields) out of bibliographic references (either given as plain-text or as a document image) – it is the part of the system with the highest degree of significance. Only an accurately functioning information extraction module permits the integration of meta-data from heterogeneous reference sources leading to a strongly networked database [DTS+06].
Although the problem seems to be not that intricate in the first place a closer examination unsheathes a plethora of complications. Basically a research paper reference can be defined as an arbitrary series of subfields wherein each transition between subfields occurs upon parsing a specific separator symbol. Across different reference styles we can observe dramatic variations amongst separator symbols, spacing, subfield order and content representation which is illustrated quite well in the following overview of some common reference styles: [DTS+06]

- APA: Davenport, T., DeLong, D., & Beers, M. (1998). Successful knowledge management projects. *Sloan management review*, 39(2), 43-57.

- IEEE: [1] T. Davenport, D. DeLong and M. Beers, "Successful knowledge management projects," Sloan management review, vol. 39, no. 2, pp. 43-57, 1998.

- ACM: 1. Davenport, T., DeLong, D. and Beers, M. 1998. Successful knowledge management projects. Sloan management review, 39 (2). 43-57.

[1] defined as structured data about data [BNP99, LAWH02, Sen04]

- MISQ: Davenport, T., DeLong, D., and Beers, M. "Successful knowledge management projects," Sloan management review (39:2) 1998, pp 43-57.

- JMIS: 1.Davenport, T.; DeLong, D.; and Beers, M. Succesful knowledge management projects. Sloan management review, 39, 2 (1998), 43-57.

- ISR: Davenport, Thomas, David DeLong and Michael Beers, "Succesful knowledge management projects," Sloan management review, 39, 2, (1998), 43-57.

Systems like CiteSeer depend on the ability to assign the highly differing syntactical representations of the same paper to one semantical entity, i.e. the file containing the corresponding document, for instance to allow grouping of information from multiple citing papers, bi-directional links between citations and the generation of citation frequency statistics [LGB99b]. The vast number of varying bibliography styles and the problematics of explicitly defining each one argue for a machine learning approach and against a rule-based approach although such systems are being developed and seem to work effectively. As a major drawback of the rule-based approach should be recorded that it requires a domain expert who has to explicitly formulate his knowledge about each reference style that should be recognized by the information extraction component. Therefore each style has to be manually analyzed first and then implemented in the system as a rule, dependent on the symbols already parsed, which is error-prone and time-consuming. Furthermore the resulting system is not adaptive at all; meaning that references represented in undefined bibliography styles cannot be recognized at all - even if only a single symbol differs.
By applying machine learning techniques to the problem area we negate the need for a domain expert and produce a robust system as small changes in syntactical representations just have a restricted influence on the results, i.e. most subfields – especially those not directly adjacent to the varying one - should not be affected at all and even a completely correct classification is possible. Furthermore it is very adaptive as major changes in the problem domain (common reference styles) can be tackled by training the system on updated datasets which just requires some preprocessing in contrast to the involved definition of new rules and the prepending analysis of the changes in the reference styles which has to be done manually in a rule-based system.

## 2 Probabilistic Finite State Transducers

As the model of choice we selected probabilistic finite state transducers (PFST) which are analyzed in the following section by introducing the mathematical foundations, presenting the possible operations and the used implementation as well as finally highlighting the excellent applicability to the problem domain of bibliographic meta-data extraction.
The reader is referred to [Het06] for a brief overview about the topic as many of the principles presented therein receive a more thorough treatment in this elaboration and thus it can be used as a rough guideline for parts of this section.

## 2.1 Mathematical Introduction

In the following part of the elaboration we will formally define probabilistic finite state machines[2] (FSM) by extending the models of finite state automata with transition outputs yielding finite state transducers and finally addings weights and scoring mechanisms after supplying the required algebraic foundations.
The papers [THC05a, THC05b] give a thorough theoretical treatment of probabilisitc finite state machines and can be used in addition to the mathematical introduction given here as they explain the topic at great length. The first part [THC05a] is devoted to establish the model of probabilistic finite state machines and inspect their properties whereas the second part [THC05b] examines the relations to other string-generating devices like hidden Markov models.
[Moh97] is a very good work as well giving a clean and elaborate algebraic introduction to probabilistic finite state transducers that probably is more similar to the specifications given here than in the work mentioned above, and a detailed inspection of their properties. In addition the applicability to language and speech processing tasks is inspected and corresponding algorithms are given.

### 2.1.1 Finite State Acceptors

A finite state automaton or acceptor (FSA) is the model of which probabilistic finite state transducers are generalized from and thus we will begin with an examination of them first. Finite state automata consist of a finite, non-zero number of states $S$ and transitions $t \in S \times (\{\epsilon\} \cup \Sigma) \times S$ depending on an input symbol out of an input alphabet $\Sigma$ between those states. If the series of transitions $s_0 \rightarrow^{t_{j_0}} s_{i_0} \rightarrow^{t_{j_1}} \ldots \rightarrow^{t_{j_m}} s_{i_n}$ is well-defined for each possible input composable out of the input alphabet we speak of a deterministic finite state automaton (DFSA/DFSM); if multiple transitions can be taken at any point during the parse, i.e. there exist $t_0, t_1$ such that $t_0 = (s_i, a, s_n)$ and $t_1 = (s_i, a, s_m)$ with $n \neq m$, we speak of a non-deterministic finite state automaton (NDFSA/NDFSM). So the state transition mapping is a function in the deterministic case and a relation in the non-deterministic case.
Furthermore an initial state[3] $s_0$, where parsing begins, and a set of final states $F$, which define the accepting inputs, have to be specified. Thereby each finite state automaton defines a language which covers every word that gets accepted by the automaton. Exactly those languages that get accepted by a finite state automaton are called regular languages. If parsing reaches a situation where no mapping exists for the given input symbol and the current state we stop parsing and reject the input string. It should be noted that we can construct a deterministic finite state automaton for each non-deterministic one that accepts the same language although the result may be an exponential growth in the number of necessary states.

**Definition** A *finite state automaton* is a quintuple $(\Sigma, S, S_0, \delta, F)$ where:

[2] meaning both finite state acceptors and transducers
[3] or a set of initial states $S_0$ in the nondeterministic case respectively

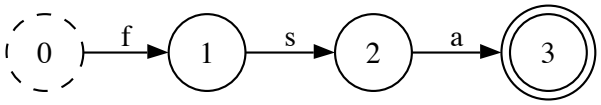
- $\Sigma$ is a finite input alphabet with $\Sigma \neq \emptyset$.
- $S$ is a finite set of states with $S \neq \emptyset$.
- $S_0$ is a set of initial states with $S_0 \subseteq S$.
- $\delta$ is a state transition relation with $\delta : S \times (\{\epsilon\} \cup \Sigma) \times S$.
- $F$ is a set of final states with $F \subseteq S$.

Each finite state automaton defines the regular language $L(\text{FSA}) = \{w|\text{FSA reaches a final state on parsing w}\}$ consisting of all words whose parses get accepted.

**Example** $\Sigma = \{f, s, a\}; S = \{0, 1, 2, 3\}; S_0 = \{0\}; F = \{3\}$

$\delta$ is a state transition function (as the given finite state acceptor is deterministic) with $\delta : S \times (\{\epsilon\} \cup \Sigma) \to S$ and takes the following values:

$\delta(0, f) = 1; \delta(1, s) = 2; \delta(2, a) = 3$
$delta(i, j)$ undefined for $(i, j) \notin \{(0, f), (1, s), (2, a)\}$




This very simple example finite state acceptor just accepts the input string "fsa" and rejects anything else.

### 2.1.2 Regular Languages

In the following part we will formalize regular languages (also called Type-3 languages in the Chomsky hierarchy [Cho56, CM58, Cho59]) and their correspondence to finite state automata for clarification issues as well as to prepare the theoretical expansion to transducers and probabilistic weighting.

**Definition** The *regular languages* over an alphabet $\Sigma$ are defined recursively as follows:

- $\emptyset$ is a regular language.
- $\{\epsilon\}$ is a regular language.
- $\forall a \in \Sigma$: $\{a\}$ is a regular language.
- $\forall$ regular languages $A, B$: $A \cup B$, $A \cdot B$, $A^*$ are regular languages.
- no other languages over $\Sigma$ are regular languages.

**Theorem 2.1** (Kleene's theorem) *Any language that gets accepted by a finite state automaton is a regular language.*
*Any regular language gets accepted by a finite state automaton.*

**Proof** Refer to [Mar02] (page 145ff.) or your theoretical computer science literature of choice as the proof is rather technical and too long to include it here. ∎

**Corollary 2.2** *All finite languages, i.e. all languages that only consist of a finite number of words, are regular languages.*

**Proof** As each singleton $\{a\}$ for $a \in \Sigma$ or $a = \epsilon$ is regular and the union/concatenation operation is closed over the collection of regular languages we can construct arbitrary subsets of words over $\Sigma$ which stay regular languages. Thereby we can build each finite language using the operations mentioned above and still maintain the property of regularity. In the case that the finite language contains no words at all it is regular by definition. ∎

Each regular language can be expressed as a regular expression and vice versa and in fact both classes share the same level of expressiveness. Furthermore operations on finite state automata can be treated as operations on the languages[4] they generate and insights gained on one model may be applied to the other. So we can check for instance the isomorphism of two given finite state automata by testing whether they both generate the same language and check the equality of two given regular languages by testing whether they both get generated by the same minimal deterministic finite state automaton respectively.
Although both models are isomorphic it should be noted that some regular languages can only be expressed by exponentially growing finite state automata while the corresponding regular expressions only grow linearly in size which is an important practical issue.

### 2.1.3 Finite State Transducers

The inclusion of a symbol of the output alphabet $\Gamma$ that is output on each transition $t \in S \times (\{\epsilon\} \cup \Sigma) \times (\{\epsilon\} \cup \Gamma) \times S$ into the model leads to a finite state transducer (FST). If $t$ only depends on the current state we speak of the Moore model; if it additionally depends on the input symbol we speak of the Mealy model. Both models are equivalent in the sense that they can be algorithmically transformed vice versa and in practice mixed models are often used as well.
Thus we do not only accept or reject a given input string any more as in the case of finite states automata but generate an output string depending on the parse of the input (or reject the input by outputting nothing). So each finite state transducer introduces a relation between input and output language. If at any point during the parsing procedure we reach a point where $\delta$ is undefined for the current input symbol and state we reject the given input string like in the case of finite state acceptors.

**Definition** A *finite state transducer* is a six-tuple $(\Sigma, \Gamma, S, S_0, \delta, F)$ where:

- $\Sigma$ is a finite input alphabet with $\Sigma \neq \emptyset$.
- $\Gamma$ is a finite output alphabet with $\Gamma \neq \emptyset$.

[4] or rather as operations on the sets of words that are contained in the corresponding language (cf. to set theoretical operations)

- $S$ is a finite set of states with $S \neq \emptyset$.
- $S_0$ is a set of initial states with $S_0 \subseteq S$.
- $\delta$ is a state transition relation with $\delta : S \times (\{\epsilon\} \cup \Sigma) \times (\{\epsilon\} \cup \Gamma) \times S$.
- $F$ is a set of final states with $F \subseteq S$.

Each finite state transducer FST computes a rational relation $\omega$ between input language $\Sigma^*$ and output language $\Gamma^*$ such that:

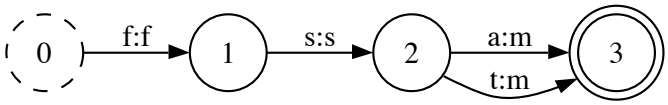
$$x \omega y \text{ if and only if } \exists s \in S_0, f \in F : (s, x, y, f) \in \delta^* \tag{1}$$

This means that FST transduces a string $x \in \Sigma^*$ out of the input language into a string $y \in \Gamma^*$ out of the output language if and only if there exists a path from an initial state $s \in S_0$ to a final state $f \in F$ with input label $x$ and output label $y$.

**Example** $\Sigma = \{f, s, a, t\}; \Gamma = \{f, s, m\}; S = \{0, 1, 2, 3\}; S_0 = \{0\}; F = \{3\}$

$\delta$ is a state transition function (as the given finite state transducer is deterministic) with $\delta : S \times (\{\epsilon\} \cup \Sigma) \times (\{\epsilon\} \cup \Gamma) \to S$ and takes the following values:

$\delta(0, f, f) = 1; \delta(1, s, s) = 2; \delta(2, a, m) = 3; \delta(2, t, m) = 3$
$\delta(i, j, k)$ undefined for $(i, j, k) \notin \{(0, f, f), (1, s, s), (2, a, m), (2, t, m)\}$




This easy example finite state transducer just relates the input strings “fsa” and “fst” to the output string “fsm” and rejects anything else.

### 2.1.4 Probabilistic Weighting

By assigning weights to transitions $w(t)$ and final states $w(s_{f_i})$ as well as providing an appropriate evaluation scheme (semiring $(\mathbf{0}, \mathbf{1}, \oplus, \otimes)$) we extend the given model of finite state transducers to weighted finite state transducers (WFST). If we additionally use a specific semiring for scoring such that the weight evaluation coheres to probabilistic laws we may explicitly speak of probabilistic finite state transducers which were our model of choice for the bibliographic meta-data extraction project.
Let us continue with the formal introduction of the required algebraic structures that are used in the process of score evaluation in weighted or probabilistic finite state transducers.

**Definition** A *monoid* $(M, \odot, e)$ is a set M with a binary operation $\odot : M \times M \to M$ such that:

- $\odot$ is associative, i.e. $\forall a, b, c \in M\colon\ (a \odot b) \odot c = a \odot (b \odot c)$.
- $\exists$ identity element $e$ regarding $\odot$, i.e. $\exists e \in M \forall a \in M\colon\ a \odot e = e \odot a = a$.
- $M$ is closed regarding $\odot$, i.e. $\forall a, b \in M\colon\ a \odot b \in M$.

It is called a commutative or abelian monoid if $\odot$ is commutative, i.e. $\forall a, b \in M\colon\ a \odot b = b \odot a$.

**Definition** A *semiring* $(R, \oplus, \otimes, \mathbf{0}, \mathbf{1})$ is a set R and two binary operations $\oplus$ (addition) and $\otimes$ (multiplication) satisfying the following properties:

- $(R, \oplus, \mathbf{0})$ is a commutative monoid with identity element $\mathbf{0}$.
- $(R, \otimes, \mathbf{1})$ is a monoid with identity element $\mathbf{1}$.
- Distributivity:
  $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$
  $(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$
- Annihilation:
  $0 \otimes a = a \otimes 0 = 0$

It is called a commutative semiring if $\otimes$ is commutative, i.e. $\forall a, b \in R\colon\ a \otimes b = b \otimes a$.
It is called an idempotent semiring if $\oplus$ is idempotent, i.e. $\forall a \in R\colon\ a \oplus a = a$.

**Definition** A *weighted finite state transducer* is an eight-tuple $(\Sigma, \Gamma, S, S_0, \delta, F, w, (\mathbb{R}, \oplus, \otimes, \mathbf{0}, \mathbf{1}))$ such that:

- $\Sigma$ is a finite input alphabet with $\Sigma \neq \emptyset$.
- $\Gamma$ is a finite output alphabet with $\Gamma \neq \emptyset$.
- $S$ is a finite set of states with $S \neq \emptyset$.
- $S_0$ is a set of initial states with $S_0 \subseteq S$.
- $\delta$ is a state transition relation with $\delta : S \times (\{\epsilon\} \cup \Sigma) \times (\{\epsilon\} \cup \Gamma) \times S$.
- $F$ is a set of final states with $F \subseteq S$.
- $w$ is a weight relation with $w : T \cup F \times \mathbb{R}$ where $T = \{t | t \in \delta\}$.
- $(R, \oplus, \otimes, \mathbf{0}, \mathbf{1})$ is a weight semiring where the following holds:
  $\mathbf{0} \oplus x = x$, $\mathbf{1} \otimes x = x$, $\mathbf{0} \otimes x = \mathbf{0}$, $\mathbf{0} \otimes \mathbf{1} = \mathbf{0}$, whereby
  $\mathbf{0} \in R$ identity element regarding $\oplus$ and
  $\mathbf{1} \in R$ identity element regarding $\otimes$.

Parallel transition weights are evaluated according to the $\oplus$ operator.
Serial transition weights are evaluated according to the $\otimes$ operator.

Now our model does not only establish a relation $\omega$ between input alphabet $\Sigma^*$ and output alphabet $\Gamma^*$ but it additionally allows us to compute the weight for each element $(x, y) \in \Sigma^* \times \Gamma^*$ via score evaluation according to the specified semiring. Thereby $w$ is a function if one transition only has one weight assigned to it which is the practically relevant case.

**Definition** A *probabilistic finite state transducer* is a weighted finite state transducer $(\Sigma, \Gamma, S, s_0, \delta, F, w, r)$ with a specific type of weight semiring $r \in \{r_{pr}, r_{log}\}$:

- The weight semiring $r_{pr} = (\mathbb{R}, +, \times, 0, 1)$ leads to a probabilistic evaluation of weights, i.e. we sum over parallel transition weights and multiply over serial transition weights.

- The weight semiring $r_{log} = (\mathbb{R}, min, +, \infty, 0)$ leads to a weight evaluation according to $-log$ probability, i.e. we take the minimum over parallel transition weights and sum over serial transition weights.

This specializes the model of weighted finite state transducers in the sense that the weights computed for each element $(x, y) \in \Sigma^* \times \Gamma^*$ now represent the corresponding probablities or $-log$ probabilities respectively which allows us to retrieve the most probable mapping for a given input string via shortest path search (i.e. Viterbi or $A^*$).
Using $-log$ probability reduces the operations "by one level", i.e. multiplication to addition and addition to taking the minimum, thus being the practically relevant solution due to a significant reduction of computational complexity.

**Example** Here we want to illustrate the different evaluation procedures depending on the type of semiring chosen as presented in [Het06]. We chose this example as it explains the different scoring mechanisms comprehensible although it is not practically relevant due to the nondeterministic and in fact meaningless model after all. Let us first take a look at the weighted model without any score evaluations that have been applied yet.

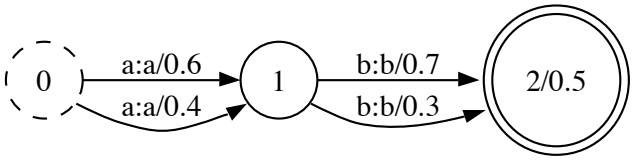
$\Sigma = \{a, b\}; \Gamma = \{a, b\}; S = \{0, 1, 2\}; S_0 = \{0\}; F = \{2\}$

$\delta$ is a state transition function (as the given weighted finite state transducer is deterministic) with $\delta : S \times (\{\epsilon\} \cup \Sigma) \times (\{\epsilon\} \cup \Gamma) \to S$ and takes the following values:

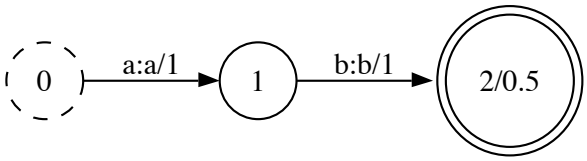
$\delta(0, a, a) = 1; \delta(1, b, b) = 2$
$\delta(i, j, k)$ undefined for $(i, j, k) \notin \{(0, a, a), (1, b, b)\}$

$w$ is a weight relation (as both transitions have two weights associated with them) with $w : T \cup F \times \mathbb{R}$ where $T = \{t | t \in \delta\}$ and spans the following elements:

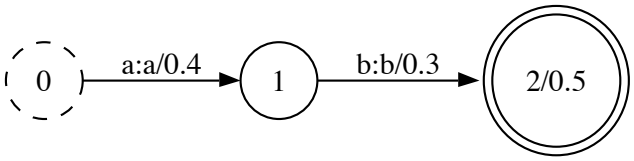
$w = \{((0, a, a), 0.6), ((0, a, a), 0.4), ((1, b, b), 0.7), ((1, b, b), 0.3), (2, 0.5)\}$




A weight evaluation according to the semiring $r_{pr} = (\mathbb{R}, +, \times, 0, 1)$, i.e. a probabilistic evaluation of weights, relates the input "ab" to the output "ab" while giving it a score of 0.5 ($= 1 \times 1 \times 0.5$).




A weight evaluation according to the semiring $r_{log} = (\mathbb{R}, min, +, \infty, 0)$, i.e. a weight evaluation according to $-log$ probability, maps the input string "ab" to the output "ab" scoring it with 1.2 ($= 0.4 + 0.3 + 0.5$).




## 2.2 Operations

Here we will illustrate and explain the various mechanisms that can be applied to finite state transducers for modelling or optimization purposes. The task of constructive operations is to easily derive models with new properties and capabilities out of given models (extension) whereas identity operations try to reduce or determinize given models without changing their properties (optimization).

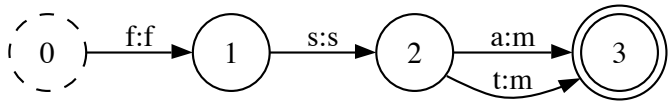
### 2.2.1 Constructive Operations

Each constructive operation has the capability to generate a new model with distinct properties from the source model(s) by applying a series of defined modifications, i.e. applying an algorithm, on the source model(s). By using an adequate series of constructive operations on some simple machines we generate a complex one with the desired capabilities step by step and this with a high level of flexibility due to the modularity of the process leading to a fast and intuitive modelling procedure.
The set of constructive operations that can be applied to finite state machines spans the computation of a closure, the union of a group of machines, the concatenation of machines, the complementation of a machine, the intersection of machines and the composition of

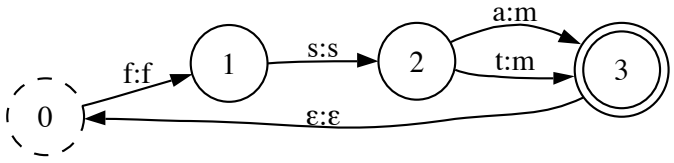
machines. We will now continue and explain each operation on its own and give corresponding examples whereby the reader may provide himself[5] an overview in [Het06].

Computing the transitive closure of a finite state machine $M$ is done by inserting epsilon transitions from all final states $F$ of $M$ to all starting states $S_0$ of $M$ thus generating the language $L(M_+) = \{w_1 \dots w_i | i > 0, w_j \in L(M), 1 \leq j \leq i\}$ which consists of an arbitrary ($> 0$) number of words out of $L(M)$. Addition of the reflexive closure allows for parsing to stop in the starting states as well, i.e. $F_{M_*} = F_M \cup S_{0_M}$, which leads to the language $L(M_*) = L(M_+) \cup \{\epsilon\}$.

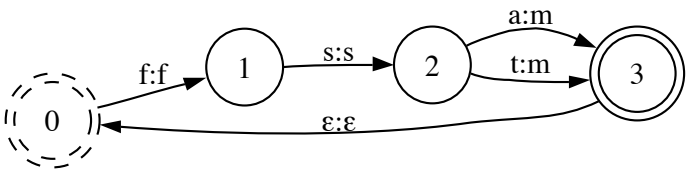
**Example** (Closure)




source machine




transitive closure



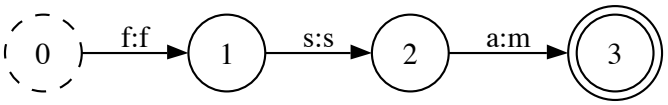

transitive and reflexive closure

The union of a set of finite state machines $M_1, \dots, M_n$ is interpretable as the union of the languages $L(M_1), \dots, L(M_n)$ they generate which results in a machine $Q$ that accepts

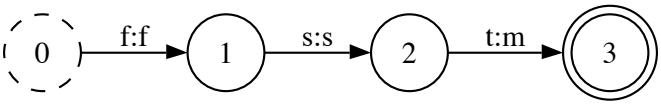
[5] apologies to any female readers but we will restrict ourselves to the masculine form for the sake of simplicity

everything that is accepted by any machine of the set but nothing else, i.e. $L(Q) = L(M_1) \cup \ldots \cup L(M_n)$. Technically we just introduce a new starting state $q_0$ from where epsilon transitions are leading to all original starting states $s_{0,j,M_i} (i \in 1, \ldots, n; j \in 1, \ldots, |S_{0,M_i}|)$ of the source machines whereby the freshly introduced state is the only starting state any more, i.e. $S_0 = \{q_0\}$.

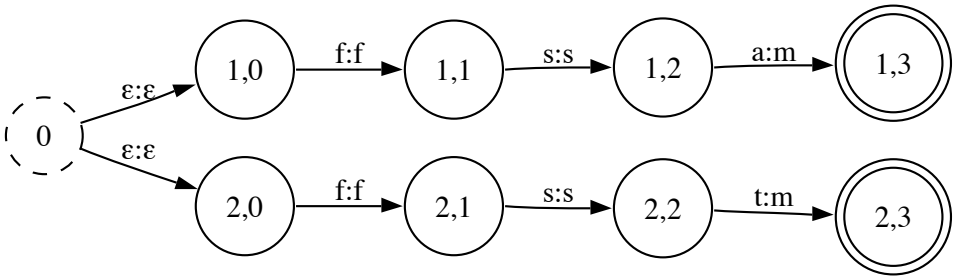
**Example** (Union)




source machine 1



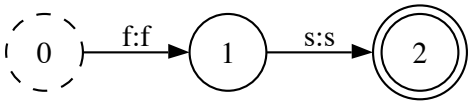

source machine 2



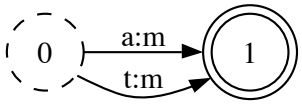

union of source machines

Concatenation can be understood as serially joining the languages $L(M_1), \ldots, L(M_n)$ generated by the involved finite state machines $M_1, \ldots, M_n$ thus leading to a machine $Q$ with $L(Q) = L(M_1) \bullet \ldots \bullet L(M_n)$ that accepts all words $w = w_1 \ldots w_n$ with $w_1 \in L(M_1), \ldots, w_n \in L(M_n)$. Furthermore the starting states of the first machine are the starting states of the resulting model, i.e. $S_0 = S_{0,M_1}$, and the final states of the last machine are the final states of the resulting model, i.e. $F = F_{M_n}$, whereby distinct machines get connected via an epsilon transition.

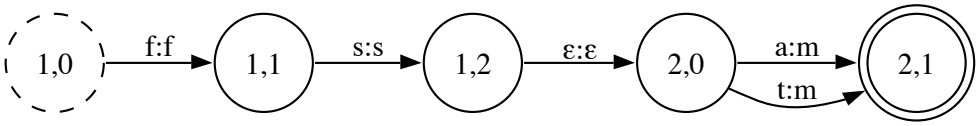
**Example** (Concatenation)




source machine 1




source machine 2

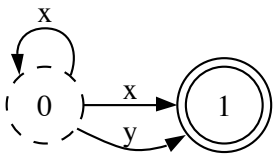



concatenation of source machines

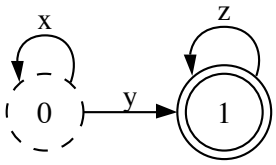
The complementation[6] of a finite state automaton $A$ yields a model $A_{compl}$ that shares the input and output alphabet with $A$ but accepts exactly those words that are rejected by the original model, i.e. $L(A_{compl}) = \{w|w \notin L(A)\}$. No example is given for this operation as illustration is a bit problematic and not really helpful at all.

By computing the intersection[7] of finite state automata $A_1, A_2$ we generate a model $A$ in which the target states $(p, q)$ are derived from source state pairs $p$ out of $A_1$ and $q$ out of $A_2$. Target states $(p, q)$ are final if $p$ and $q$ are both final states in their source automata and transitions with a label $l$ are included if $p$ and $q$ have a corresponding transition with label $l$. In the case of weighted finite state automata we combine the source weights according to $\otimes$.

**Example** (Intersection) [Het06]



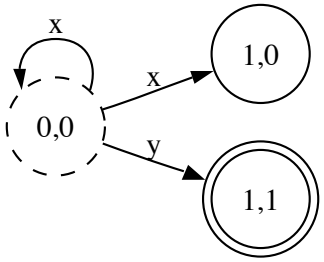

source machine 1




[6] we can only apply the complement to finite state automata

[7] the intersection operation can only be applied to finite state automata

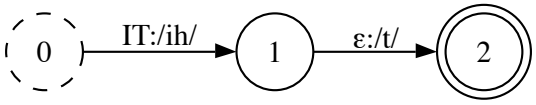
source machine 2



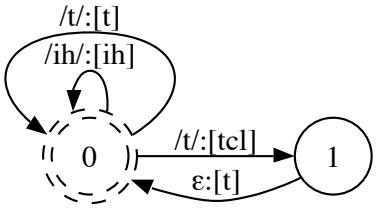

intersection of source machines

Via composing[8] two final state transducers $T_1, T_2$ we construct a model $T$ where the target states $(p, q)$ are derived from source state pairs $p$ out of $T_1$ and $q$ out of $T_2$. $(p, q)$ is final if both $p$ and $q$ are final and weight evaluation is done according to $\otimes$. Differing from the intersection is now that transitions labelled $l : m$ are included if in $p$ exists a transition with label $x : n$ and in $q$ exists a transition with label $n : y$, i.e. we compose the transition labels.

**Example** (Composition)
This example is taken out of [Het06] as it visualizes the application of the composition operation for the task of natural language processing. By composing one transducer that maps words to phonemes with one which maps phonems to phones we obtain a final model where words get mapped to phones.

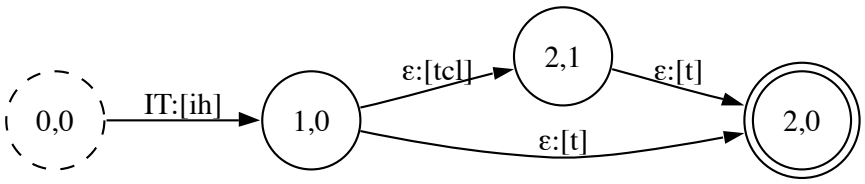



source machine 1 (words → phonemes)




source machine 2 (phonemes → phones)

[8] composing two final state automata is identical to intersecting them



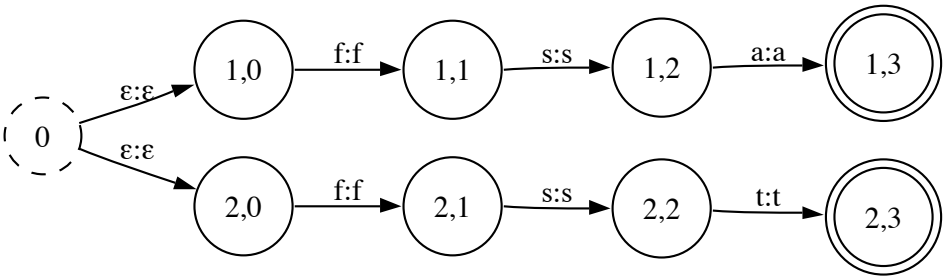

composition of source machines (words → phones)

### 2.2.2 Identity Operations

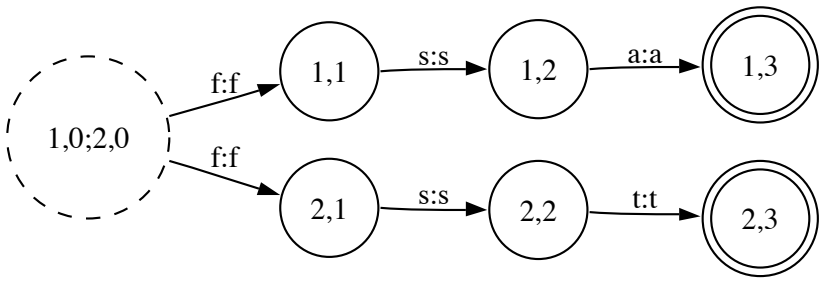
Identity operations are distinct from constructive ones as they only aim for a syntactical reduction of the model size meaning the resulting model still maintains exactly the same transition relation, weight relation and scoring mechanism as the source model, thus still relating each given input to the original output with an unchanged score, but optimally needing less states and transitions in the case of epsilon removal and minimization. Determinization on the other hand may increase the storage requirements, i.e. increase the number of states and transitions, but then reducing the complexity of the parsing procedure as only one path in the machine may match any given input sequence any more.
We will continue with a short description of each identity operation while giving examples along the way. For a very detailed treatment of the topic and corresponding algorithms the interested reader is referred to [Moh97, MPR00]. It should be noted that identity operations are always applied to only one source model as the goal is only optimization whereas some constructive operations may generate models that have been derived from multiple sources.

The first step to optimize a given model is to remove all epsilon transitions included like shown below which is technically achieved by computing the epsilon closure for each state, i.e. the set of states that is reachable by an arbitrary number of epsilon transitions, and aggregate them accordingly.

**Example** (Epsilon Removal)



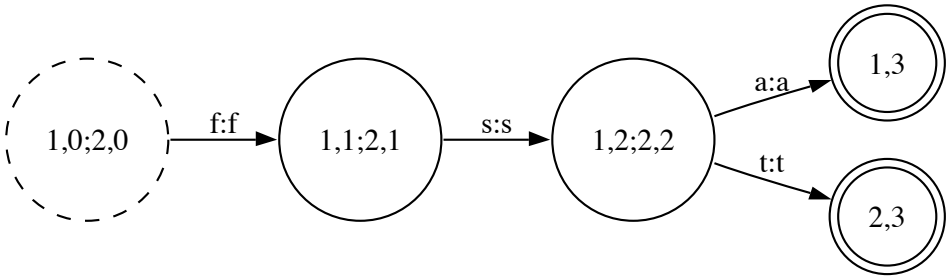

source machine




resulting machine (epsilons removed)

Now it is possible to determinize the given model which means to resolve all ambiguities by constructing target states which are derived from subsets of (input states,output,weight) and an adequate delay of outputs and weights. The output of the transitions in the target model correspond to the least common prefix of subset outputs. For finite state automata the worst case is exponential in the number of states whereas for finite state transducers the worst case may be infinite due to unresolvable ambiguities, i.e. one input sequence gets mapped to multiple output sequence.

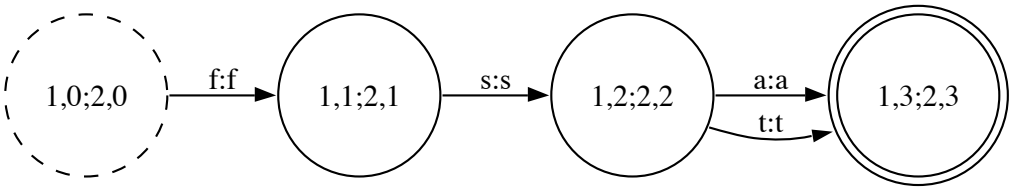
**Example** (Determinization)




resulting machine (determinized)

The task of minimization is to generate a deterministic model with a minimal number of states that is equivalent to the given one, which is achieved by merging the equivalent states. For cyclic finite state machines the complexity is $O(Nlog(N))$ whereas for acyclic ones it is linear.

**Example** (Minimization)




resulting machine (minimized)

## 2.3 Implementation

For our implementation we used the finite state transducer toolkit by the Massachussettes Institute of Technology (MIT-FST[9]) that is published as open source software under the BSD license by Lee Hetherington. A brief treatment of its design and implementation is given in [Het04] whereas in [MPR00] the finite state transducer library used by AT&T is handled in detail. The toolkit is composed of a large number of command-line tools where each one encapsulates a single functionality and works using the UNIX pipe which allows for rapid development of models via concatenated commands and/or shell scripts. Alternatively a C++ interface that incorporates exactly the same functionality but allows for easy extension and modification is provided. As our system has been designed primarily for research purposes and not for mainstream usage we chose the former solution.
Particulary the ability to generate a weighted transducer model by expectation maximization training of an unweighted transducer based on a set of pairs of input and output strings allows for efficient development of language models. A desirable feature that is not implemented in MIT-FST is the possibility to define transitions accepting all symbols out of the input alphabet that have no explicitly specified transition in the corresponding state as this would allow for a more flexible modelling in some places and especially negate the effect of non-acceptance upon the parse of undefined input symbols.

## 2.4 Applicability

Finite state transducer techniques have been successfully applied to various tasks of computational linguistics such as dictionary encoding, text processing and speech processing due to their ease of use and their affinity to regular languages. A thorough treatment of the model itself and various applications for natural language processing for the interested reader is given in [RS97] as we will restrict ourselves to the task of bibliographic reference recognition in this work. Every bibliographic reference is basically a string which is composed out of a series of different subfields such as author, title or journal whereby those subfields are separated by specific symbols such as a dot or a semicolon. The subfields are independent in the sense that their content stands in no relation to the content of other subfields which allows to model each subfield as a separate finite state transducer and link them together adequately using the knowledge of the occuring separator symbols and the class of the previous subfield.
By using finite state transducer operations we can easily build the required language model in a modularized way out of the various subfield models. So we can use the union operation on a set of subfield transducers to represent the choice between one of them, concatenate a model which parses the separator symbols and compute the transitive closure of the resulting model to allow an arbitrary (non-empty) series of different subfields to be recognized. We will inspect the construction of the language model in a more detailled way in the section about system design as we just wanted to give a first impression of the practical usability of finite state transducer models for the task area of bibiographic reference recognition.

[9] http://people.csail.mit.edu/ilh//fst

# 3 System Design

This part of the work is devoted to the bibliographic reference recognition system we constructed using probabilistic finite state transducers. Thereby the task of bibliographic reference recognition can be understood as seperating a given string (=reference) into an adequate set of subfields with the goal to extract the bibliographic meta-data and transform it into a machine-readable format (BibTeX in our case).

**Example** Bibliographic Reference Recognition (cf. Motivation)

**Input:**
Davenport, Thomas, David DeLong and Michael Beers, "Successful knowledge management projects," Sloan management review, 39, 2, (1998), 43-57.

**Output:**
author = "Davenport, Thomas and DeLong, David and Beers, Michael"
title = "Successful knowledge management projects"
journal = "Sloan management review"
volume = "39"
number = "2"
year = "1998"
pages = "43-57"

First we will present the design principles we have applied and outline the underlying development process of the system from scratch to the current status. Then the architecture of the system is inspected whereby we will differ between the language model which handles the task of tagging the subfields with appropriate handles, the front-end which serves as an graphical user interface to the system and the back-end which handles the data conversion between the front-end and the language model. In the next part of this section the used training data which served as a source for deriving the weights of the final language model via expectation maximization training is presented and the general benefits and differences in contrast to rule-based systems are discussed in detail. Finally we wil review some possible improvements of the current system and talk over their expected effects.

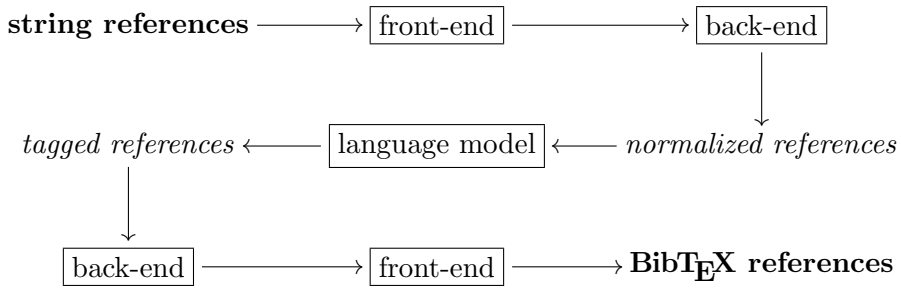
## 3.1 Modelling Approach

The first step before the system design started was to get used to the MIT-FST toolkit, understand the assortment of tools and especially how to use them efficiently. Therefore we started out with the design of a simple rule-based system that recognizes bibliographic references in the BibTeX style 'plain' whereby things like the user interface or the conversion from transducer output to BibTeX have been reused in the current system later on. During this process we built a language model for the author's name according to the

CiteSeer database and constructed models for common representations like page numbers or dates. These parts of the system have been excluded from the final design as they slowed down the runtime considerably and the experimental results have been quite good without them as well. We will discuss the possible incorporation in the current system in the subsection about possible improvements at the end of this part.
Due to the evolutionary type of design we chose the plan was to extend the functionality of the system iteratively and add new functionality stepwise. The goal was to build a style-independent probabilistic model by ongoing refinement of the prototypes with a steady performance evaluation to measure the progress. Unfortunately this restricted to hand-written test cases for quite some time as the underlying training and testing data has not been acquired until the system design matured. We started out by relaxing the field sequence and the type of seperator symbol used between the subfields, then extended the types of subfields to match the dataset and finally trained the inter-field transition probabilities according to expectation maximization training to give a basic outline of design. It is presented in detail in the following section.

## 3.2 System Architecture

In this part we will inspect the different parts of the system, present their functionality and how they communicate to each other. The basic workflow is depicted in the following diagram.




Thereby we enter a set of bibliographic references as plain-text into the front-end which passes them to the back-end where they are normalized. Then the language model takes the normalized references one by one and tags the subfields according to the finite state transducer parse. Finally the tagged output gets converted into BibTeX format and passed back to the front-end which displays the results to the user. Details of the different modules are thoroughly presented in their corresponding subsections.

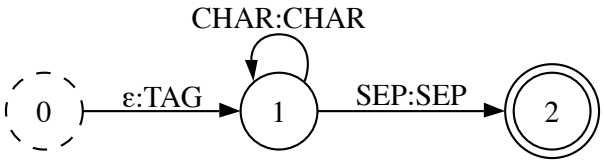
### 3.2.1 Front-End

This is the part of the system which handles all the user interaction via a PHP-based web application. Bibliographic references can be entered into a multi-line HTML text area whereby each gets passed one by one to the rest of the system for classification and the

resulting BibTeX entries are displayed on a results webpage. A planned extension is the ability to enter a document image as input, process it with OCR and pass the recognized text into the system for bibliographic meta-data extraction which would allow for easy application to scanned references. Basically this module has no functionality on its own and only serves usability purposes.

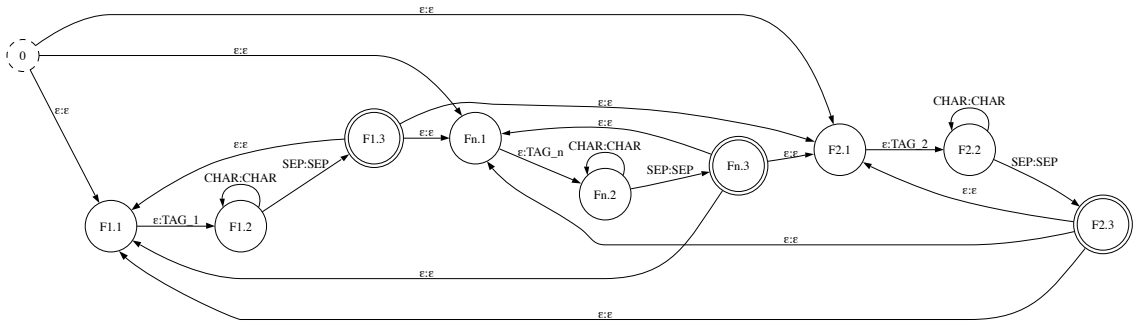
### 3.2.2 Language Model

According to the used dataset (cf. Training Aspects for details) we have specified a finite state transducer for each possibly occuring subfield – namely author, booktitle, date, editor, institution, journal, location, note, pages, publisher, tech, title, volume. These subfield transducers are structured quite simple as they only output the corresponding subfield tag, then allow the parse of an arbitrary number of unspecified symbols via an intrafield unigram and stop in a final state upon reading a separator symbol as shown in the diagram below. Thereby TAG represents the type of subfield (i.e. author), CHAR consists of the whole alphabet and SEP is a subset of CHAR including all possible interfield separator symbols like a colon or a dot. This allows for the parsing of separator symbols intrafield as we cannot exclude the feasibility of them occuring there – like a dot that indicates an abbreviation. Although the characters occuring intrafield are only incorporated using an unigram, i.e. their weights are constant for each type of subfield and do not rely on already parsed characters, the results are quite good as we will see.




simplified subfield model

Now the overall language model is built as the union of the different subfields whereby each final state gets additional epsilon transitions leading to the first states of the subfield transducers. Thus we basically model a bigram of the subfields – as it is shown below in a simplified form – which means that the parse of a subfield transducer depends on the type of the last subfield.




simplified language model

By composing the resulting language model (after training) with a finite state transducer that represents a given bibliographic reference we receive a transducer that delivers us a tagging of subfields according to the highest probabilistic evaluation.

### 3.2.3 Back-End

The back-end handles the task of data conversion between the language model and the front-end. Thereby the entered reference (plain-text) is normalized in the sense that unknown symbols are removed as they would interfere with the parsing process and symbols are mapped in such a way that allows for conversion of the string into a finite state transducer, i.e. we have to map spaces to a special symbol, convert characters with a specialized meaning[10] in MIT-FST to another representation and separate each symbol by a real space. Now the modified input is ready for finite state transducer conversion via the MIT-FST tool fst_from_string. The resulting finite state transducer gets composed with the language model and a following search on this one leads to a tagging of subfields according to the highest probability.
The tagged output of the finite state transducer parse is still in normalized format and has to be processed back into human-readable form to allow passing it back to the front-end. Thereby all conversion steps described above have basically to be reverted, i.e. symbols are mapped back to their original meaning, the newly introduced spaces are removed and the original spaces are restored. Finally each tagged sequence of the whole reference gets interpreted as a BibTeX subfield and the results are passed back to the front-end for displaying them to the user. In the current state all entries are classified as @misc as no mechanisms for differentiating between the reference types have been incorporated into the system yet.

## 3.3 Training Aspects

Here we discuss the Cora dataset[11] that has been used for training purposes and talk about the benefits of a system based on training against a rule-based one. We chose this

[10] i.e. an epsilon is denoted by a comma
[11] http://www.cs.umass.edu/ mccallum/data/cora-ie.tar.gz

dataset as it seems to be the most widespread one that has been used for the training and evaluation of bibliograhic reference recognition systems and it is freely downloadable from the web as well.

### 3.3.1 Dataset

We partitioned the 500 research paper citations included in the Cora dataset into a subset of training and testing samples whereby we just took the first 350 for training and the following 150 for testing as unfortunately no specific partitioning has been published for comparison purposes by other researchers [DTS+06, MNRS99a, PM04] and we were unable to retrieve the explicit partitioning used by them. To allow for an easy measurement of performance it would make sense to enforce a specific partitioning across the different systems as changes in the training and testing set may very well lead to significant changes in accuracy thus obfuscating the results.
The references contain exactly those 13 fields that have been modelled as subfields and have been listed in the subsection about the language model. By application of expectation maximization training on the language model described above using the training subset of the Cora references we acquire our final probabilistically weighted language model. Thereby the field content is trained as a unigram and the field sequence is trained as a bigram because of the structure of the language model.

### 3.3.2 Benefits

By designing a system that is based on training data we negate the need for a domain expert that manually analyzes the different BibTeX styles and derives adequate rules out of them. This reduces the time effort needed for defining the rules and prevents errors during the process thus resulting in a highly efficient system. Furthermore we may adjust the language model to new reference styles by repeating the expectation maximization training process on a new dataset that fits those styles. Therefore basically no manual intervention is needed as the training procedure is highly automated. This is not possible in rule-based systems as another manual analysis of the new styles and an adequate rule definition has to be done again. Hence we can note that our system works with a high degree of flexibility and adaptability in regard to changing bibliographic reference styles.
Another benefit is the resulting robustness of the system as small changes in the underlying styles should not influence the overall results too much as it may be the case in rule-based systems depending on the quality of the rule definition. This means that i.e. a switch of a separator symbol or a change of field ordering could still deliver the correct result but should at worst result in a local error due to the structure of our language model as subfields are only influenced by their direct predecessor. It is thinkable but unprobable that such small changes propagate errors through the whole reference but when using a rule-based approach all those eventualities have to be incorporated in the rules or the parsing cannot be successful at all. However we would have to conduct more experiments to support the statement.

### 3.4 Self-Evident Improvements

Let us look at some obvious methods for improving the system further in this section. The first thing to think about is to exchange the subfield content unigram with a bigram or trigram as it is quite easy to implement and could increase the system accuracy in a significant way. It would also be possible to construct the language model as an interfield trigram thus including the last two predecessors of each subfield in the decision process. Although these measures will improve the system very probably it comes with the side-effect of performance reduction. Therefore we would need to find a good trade-off between runtime and accuracy as the system needs to be practically usable by conducting further experiments.
Another evident modification would be to incorporate the already partly available models for common representations like an authors' name or date model thus resulting in some type of hybrid system as this could already be seen as a rule definition which restricts the system's robustness but improves results as long as the common representations' models match the corresponding parts of the reference strings. Further experiments with different setups of included representations would have to be run to determine an optimal language model in respect to the testing data. But it should be regarded that inclusion of too many specific models may very well lead to better results on the given set of testing samples but reduce the system's performance in general.
Finally it would be nice to implement some sort of reference type classification as all BibTeX entries are currently classified as @misc only but the problem is not as trivial as it may sound in the first place. For instance it is not easy to distinguish between @journal or @inproceedings as both may share a relatively identical structure. Probably the simplest way to include a type differentation would seem to build a classificator that distinguishes after the overall transducer parsing process depending on the occuring subfields and their content what kind of reference we encountered.

## 4 Performance Evaluation

Here we will inspect the performance of the system after defining relevant performance measures that serve as a basis for evaluation and allow for comparison. After presenting the other existing relevant systems in the area of bibliographic meta-data extraction we will compare the experimental results and conclude the section with an analysis and a dIscussion of possible measures to improve them further.

### 4.1 Performance Measures

We used the common performance measures as introduced in [PM04, DTS+06]. Thereby we have not implemented the F1-measure yet but we will give its definition as an incorporation in the near future for a better comparability of accuracy is desirable.

$$\text{word accuracy} = \frac{|\text{correctly recognized words}|}{|\text{words}|}$$

$$\text{field accuracy} = \frac{|\text{correctly recognized subfields}|}{|\text{subfields}|}$$

$$\text{F1 accuracy} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

$$\text{instance accuracy} = \frac{|\text{correctly recognized references}|}{|\text{references}|}$$

The different types of error measurements are structured in a hierarchical sense to give a better impression of the system performance. So word accuracy favors fields with many words, field accuracy favors fields with few words and instance accuracy gives a picture of how well the system performs in an overall view as only completely correct references are counted.

### 4.2 Experimental Results

Let us begin this subsection by giving a table with an overview of the different systems that have been evaluated on the Cora reference dataset and then proceed with a discussion of the results.

| system / accuracy measure | word | field | F1 | instance |
|---|---|---|---|---|
| CRF-based [PM04] | 95.4% | – | 91.5% | 77.3% |
| our system | 88.5% | 82.6% | – | 42.7% |
| HMM-based [MNRS99b] | 85.1% | – | 77.6% | 10.0% |
| rule-based [DTS$^+$06] | – | 73.34% | – | – |

We can see that the CRF-based system by [PM04] still maintains the best performance on the Cora dataset in regard to all different accuracy measurements. Especially the instance accuracy which only matches completely correct references reaches a very high value in comparison to the other systems. Our approach performs slightly worse on the word accuracy and significantly worse on the instance accuracy in comparison to that system.
The HMM-based system by [MNRS99b] again performs slightly worse on the word accuracy and significantly worse on the instance accuracy than our system. Interpretation of the strongly differing instance accuracy alongside the nearly identical word accuracy leads to the assumption that the HMM-based approach yields a lot of minor errors that only influence few words but degrade the correctness of the references overall.
As the rule-based approach by [DTS$^+$06] only measured the field specific performance no exact comparison is possible but a lower performance than our system seems probable due to the significant accuracy difference. Thus our system currently seems to be the second best performing system in regard to the Cora dataset that is available at the moment.

## 5 Acknowledgements

Thanks to Hagen Kaprykowsky and Daniel Keysers for their help on several upcoming technical issues, the preparation of relevant literature, tools and datasets as well as for the

fruitful discussions about various design issues of the resulting system which saved a lot of effort and majorly contributed to the good results.

# 6 Conclusion

As finite state transducers have proven their applicability to many tasks of computational linguistics and permit easy modelling they seem a natural choice for deriving a language model adjusted to the recognition of bibliographic references. The modularized composition of the overall language model allows for subfield model exchanges with minor efforts and the highly automated training procedure increases the reusability of our system regarding changes in reference styles and types. Especially in comparison to rule-based approaches our system yields a higher degree of robustness as no strict rulings are enforced and local errors should not propagate through the entire reference. Furthermore we negate the need for a domain expert and thus decrease the time requirements for adapting the system to changing requirements. As normally rules would have to be manually derived after analysis of the various reference styles we just have to rerun the expectation maximization training procedure on an adequate dataset representing the syntactical differences of the styles.
The relatively simple structure of the language model indicates that our system's performance can be increased significantly by replacing the intrafield unigram with a bi- or trigram and eventually the interfield bigram with a trigram. Also the models for common representations that have been built during the first development stages (i.e. authors' name model out of CiteSeer data) could be additionally incorporated for further system specialization according to a set of reference styles. Thereby we have to find an acceptable tradeoff between the system's runtime and accuracy as huge language models increase the parsing time significantly which on the one hand leads to an error rate improvement but on the other hand negates the practical usability. Additionally we want to avoid an overfitting of the system on a specific type of reference style to maintain its robustness.

## References

[BNP99] Burnett, Kathleen ; Ng, Kwong B. ; Park, Soyeon: A comparison of the two traditions of metadata development. In: *J. Am. Soc. Inf. Sci.* 50 (1999), Nr. 13, S. 1209–1217. – ISSN 0002–8231

[Cho56] Chomsky, Noam: Three Models for the Description of Language. In: *IRA Transactions on Information Theory* 2 (1956), Nr. 3, S. 113–124

[Cho59] Chomsky, Noam: On Certain Formal Properties of Grammars. In: *Information and Control* 2 (1959), Nr. 2, S. 137–167

[CM58] Chomsky, Noam ; Miller, George A.: Finite State Languages. In: *Information and Control* 1 (1958), Nr. 2, S. 91–112

[DTS+06] Day, Min-Yuh ; Tsai, Richard Tzong-Han ; Sung, Cheng-Lung ; Hsieh, Chiu-Chen ; Lee, Cheng-Wei ; Wu, Shih-Hung ; Wu, Kun-Pin ; Ong, Chorng-Shyong ; Hsu, Wen-Lian: *Reference Metadata Extraction Using a Hierarchical Knowledge Representation Framework.* http://www.iis.sinica.edu.tw/IASL/webpdf/paper-2006-Reference_Metadata_Extraction_Using_a_Hierarchical_Knowledge_Representation_Framework.pdf, December 2006

[GBL98] Giles, C. L. ; Bollacker, Kurt D. ; Lawrence, Steve: CiteSeer: an automatic citation indexing system. In: *DL '98: Proceedings of the third ACM conference on Digital libraries.* New York, NY, USA : ACM Press, 1998. – ISBN 0–89791–965–3, S. 89–98

[GMLG01] Goodrum, Abby A. ; McCain, Katherine W. ; Lawrence, Steve ; Giles, C.Lee: Scholarly publishing in the Internet age: A citation analysis of computer science literature. In: *Inf. Process. Manage.* 37 (2001), Nr. 5, S. 661–675

[Het04] Hetherington, Lee: The MIT Finite-State Transducer Toolkit for Speech and Language Processing. In: *INTERSPEECH 2004 - ICSLP: Proceedings of the Eigth International Conference on Spoken Language Processing.* Jeju Island, Korea, October 2004, S. 2609–2612

[Het06] Hetherington, Lee: *Finite-State Techniques for Speech Recognition.* http://ocw.mit.edu/NR/rdonlyres/Electrical-Engineering-and-Computer-Science/6-345Automatic-Speech-RecognitionSpring2003/3265C9CB-0CBA-477A-A959-50C3D5B8646D/0/lecture17.pdf, December 2006

[LAWH02] Lawrence A. West, Jr. ; Hess, Traci J.: Metadata as a knowledge management tool: supporting intelligent agent and end user access to spatial data. In: *Decis. Support Syst.* 32 (2002), Nr. 3, S. 247–264. – ISSN 0167–9236

[LGB99a] Lawrence, Steve ; Giles, C. L. ; Bollacker, Kurt: Digital Libraries and Autonomous Citation Indexing. In: *Computer* 32 (1999), Nr. 6, S. 67–71. – ISSN 0018–9162

[LGB99b] Lawrence, Steve ; Giles, C. L. ; Bollacker, Kurt D.: Autonomous citation matching. In: *AGENTS '99: Proceedings of the third annual conference on Autonomous Agents*. New York, NY, USA : ACM Press, 1999. – ISBN 1–58113–066–X, S. 392–393

[Mar02] Martin, John C.: *Introduction to Languages and the Theory of Computation*. McGraw-Hill Professional, 2002. – 560 S. – ISBN 0072322004

[MNRS99a] McCallum, Andrew ; Nigam, Kamal ; Rennie, Jason ; Seymore, Kristie: Building Domain-Specific Search Engines with Machine Learning Techniques. In: *AAAI Spring Symposium on Intelligent Agents in Cyberspace 1999*, 1999

[MNRS99b] McCallum, Andrew ; Nigam, Kamal ; Rennie, Jason ; Seymore, Kristie: A Machine Learning Approach to Building Domain-Specific Search Engines. In: *IJCAI '99: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1999. – ISBN 1–55860–613–0, S. 662–667

[Moh97] Mohri, Mehryar: Finite-state transducers in language and speech processing. In: *Comput. Linguist.* 23 (1997), Nr. 2, S. 269–311. – ISSN 0891–2017

[MPR00] Mohri, Mehryar ; Pereira, Fernando ; Riley, Michael: The design principles of a weighted finite-state transducer library. In: *Theor. Comput. Sci.* 231 (2000), Nr. 1, S. 17–32. – ISSN 0304–3975

[PM04] Peng, Fuchun ; McCallum, Andrew: Accurate Information Extraction from Research Papers using Conditional Random Fields. In: *HLT-NAACL*, 2004, S. 329–336

[RS97] Roche, Emmanuel ; Schabes, Yves: *Finite-State Language Processing*. Cambridge, MA, USA : MIT Press, 1997. – ISBN 0–262–18182–7

[Sen04] Sen, Arun: Metadata management: past, present and future. In: *Decis. Support Syst.* 37 (2004), Nr. 1, S. 151–173. – ISSN 0167–9236

[THC05a] Thollard, Franck ; Higuera, Colin de l. ; Carrasco, Rafael C.: Probabilistic Finite-State Machines-Part I. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 27 (2005), Nr. 7, S. 1013–1025. – ISSN 0162–8828. – Member-Enrique Vidal and Member-Francisco Casacuberta

[THC05b] Thollard, Frank ; Higuera, Colin de l. ; Carrasco, Rafael C.: Probabilistic Finite-State Machines-Part II. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 27 (2005), Nr. 7, S. 1026–1039. – ISSN 0162–8828. – Member-Enrique Vidal and Member-Francisco Casacuberta