SPPC

Shallow Processing Production Center

Jakub Piskorski
Introduction

**SPPC** - SHALLOW PROCESSING PRODUCTION CENTER

- System for shallow analysis of German free text documents (toolset of integrated shallow components)
- based on shallow text processor of SMES (G. Neumann)
- exhaustive use of finite-state technology:
  - DFKI Finite-State Machine Toolkit
    - dynamic tries
- high linguistic coverage
- very good performance
EXAMPLE: rund 60 bis 70 Prozent der Steigerungsraten
(about 60 to 70 percent increase)

- rund: lowercase
- 60: two-digit-integer

- Steigerungsraten: steigerung+[s]+rate
- bis: prep|adv

- bis: adv

- rund 60 bis 70 Prozent: percentage-NP

- rund 60 bis 70 Prozent: NP
- der Steigerungsraten: NP
Tokenizer

• The goal of the TOKENIZER is to:
  - map sequences of consecutive characters into word-like units (tokens)
  - identify the type of each token disregarding the context
  - performing word segmentation when necessary (e.g., splitting contractions into multiple tokens if necessary)

• overall more than 50 classes (proved to simplify processing on higher stages)

  - NUMBER_WORD_COMPOUND ("69er")
  - ABBREVIATION (CANDIDATE_FOR_ABBREVIATION),
  - COMPLEX_COMPOUND_FIRST_CAPITAL ("AT&T-Chief")
  - COMPLEX_COMPOUND_FIRST_LOWER_DASH ("d‘Italia-Chefs-“)

• represented as single WFSA (406 KB)
Lexical Processor

• Tasks of the LEXICAL PROCESSOR:
  - retrieval of lexical information
  - recognition of compounds („Autoradiozubehör“ - car-radio equipment)
  - hyphen coordination („Leder-, Glas-, Holz- und Kunststoffbranche“ leather, glass, wooden and synthetic materials industry)

• lexicon contains currently more than 700 000 German full-form words (tries)

• each reading represented as triple <STEM, INFLECTION, POS>

example: „wagen“ (to dare vs. a car)

STEM: „wag“
INFL: (GENDER: m, CASE: nom, NUMBER: sg)
  (GENDER: m, CASE: akk, NUMBER: sg)
  (GENDER: m, CASE: dat, NUMBER: sg)
  (GENDER: m, CASE: nom, NUMBER: pl)
  (GENDER: m, CASE: akk, NUMBER: pl)
  (GENDER: m, CASE: dat, NUMBER: pl)
  (GENDER: m, CASE: gen, NUMBER: pl)
POS: noun

STEM: „wag“
INFL: (FORM: infin)
  (TENSE: pres, PERSON: anrede, NUMBER: sg)
  (TENSE: pres, PERSON: anrede, NUMBER: pl)
  (TENSE: pres, PERSON: 1, NUMBER: pl)
  (TENSE: pres, PERSON: 3, NUMBER: pl)
  (TENSE: subjunct-1, PERSON: anrede, NUMBER: sg)
  (TENSE: subjunct-1, PERSON: anrede, NUMBER: pl)
  (TENSE: subjunct-1, PERSON: 1, NUMBER: pl)
  (TENSE: subjunct-1, PERSON: 3, NUMBER: pl)
  (FORM: imp, PERSON: anrede)
POS: noun
Part-of-Speech Filtering

• The task of POS FILTER is to filter out unplausible readings of ambiguous word forms

• large amount of German word forms are ambiguous (20% in test corpus)

• contextual filtering rules (ca. 100)
  - example: „Sie bekannten, die bekannten Bilder gestohlen zu haben“
    They confessed they have stolen the famous pictures

  „bekannten“ - to confess vs. famous

FILTERING RULE: if the previous word form is determiner and the next word form is a noun then filter out the verb reading of the current word form

• supplementary rules determined by Brill's tagger in order to achieve broader coverage

• rules represented as FSTs, hard-coded rules (filtering out rare readings)
Named Entity Recognition

• The task of NAMED ENTITY FINDER is the identification of:
  - entities: organizations, persons, locations
  - temporal expressions: time, date
  - quantities: monetary values, percentages, numbers

• identification of named entities in two steps:
  - recognition patterns expressed as WFSA are used to identify phrases containing potential candidates for named entities (longest match strategy)
  - additional constraints (depending on the type of candidate) are used for validating the candidates

• on-line base lexicon for geographical names, first names (persons)
Named Entity Recognition

• since named entities may appear without designators (companies, persons) a dynamic lexicon for storing such named entities is used

• candidates for named entities:

  example:

  Da flüchten sich die einen ins Ausland, wie etwa der Münchner Strickwarenhersteller **März GmbH** oder der badische Strumpffabrikant Arlington Socks, GmbH. Ab kommendem Jahr strickt **März** knapp drei Viertel seiner Produktion in Ungarn.

• partial reference resolution (acronyms)

• resolution of type ambiguity using the dynamic lexicon:

  example:

  if an expression can be a person name or company name (**Martin Marietta Corp.**) then use type of last entry inserted into dynamic lexicon for making decision
Phrase Recognition

• The task of FRAGMENT RECOGNIZER is to recognize nominal and prepositional phrases and verb groups

  example: [NP Das Unternehmen] [ORG Merck]] [V geht] [DATE im Herbst 1999] [PP mit einem Viertel seines Kapitals] [PP an die [ORG Frankfurter Börse]] „In autumn 1999 the company Merck will go public to the Frankfurt Stock Exchange with one fourth of it’s capital “

• extraction patterns expressed as WFSAs (mainly based on POS information and named entity type)

• phrasal grammars only consider continuous substrings (recognition of verb groups is partial)

  example: „Gestern [ist] der Linguist vom neuen Manager [entlassen worden].“
  
  Yesterday, the linguist [has] [been fired] by the new manager.

Sentence Boundary Detection

• few simple contextual rules are sufficient since named entity recognition is performed earlier
Text chart

- each component outputs the resulting structures uniformly as feature value structures (token items, lexical items, named-entity items, phrase items), together with its type and corresponding start and end positions of the spanned input expressions.

- all partial results are stored in the knowledge pool, where data computed by different components are interlinked (text chart).

- Advantages:
  - unnecessary re-computations are avoided
  - easy navigation through the space of extracted information
  - rich contextual information is provided in case of disambiguation or handling of unknown constructions
Evaluation & Performance

Performance:

INPUT: corpus of German business magazine „Wirtschaftswoche“ (1,2MB)
TIME: ~15sec. ( ~13400 wrds/sec) on Pentium III, 850MHz, 256 RAM
RECOGNIZED: 197118 tokens, 11575 named entities, 64839 Phrases

Evaluation:

<table>
<thead>
<tr>
<th></th>
<th>Recall</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPOUND ANALYSIS</td>
<td>98.53%</td>
<td>99.29%</td>
</tr>
<tr>
<td>POS-FILTERING</td>
<td>74.50%</td>
<td>96.36%</td>
</tr>
<tr>
<td>NE-RECOGNITION</td>
<td>85%</td>
<td>95.77%</td>
</tr>
<tr>
<td>FRAGMENTS (NPs,PPs)</td>
<td>76.11%</td>
<td>91.94%</td>
</tr>
</tbody>
</table>

Availability: C++ Library for UNIX, Windows 98/NT, Demo with GUI for Windows 98/NT
Future work

- multilingual shallow text processing (MSPPC)
- tokenizer with multiple classification
- early chunking
- topological parser
- more flexibility (grammar writing)

fine-grained modelling of shallow text processing into clear-cut modules

„Software LEGO“
DFKI Finite-State Tools

Why finite-State Technology?

• most of the relevant local phenomena can be easily expressed as finite-state devices
• time & space efficiency
• interesting closure properties
• existence of efficient determinization, minimization and optimization transformations
• there exist much more powerful formalisms like context free grammars or unification grammars, but industry prefers more pragmatic solutions
• finite-state technology is recently in the centre of attention

Why own finite-state tools?

• current finite-state packages: only binary code available
• new algorithms relevant to shallow processing, not provided by current tools
DFKI Finite-State Tools Overview

- Library of tools for building, combining and optimizing finite-state machines (FSM)

- FSM - Finite-State Machine
generalization of:

  - FSA
    finite-state automata

  - WFSA
    weighted finite-state automata

  - FST
    finite-state transducers

  - WFST
    weighted finite-state transducers

- arbitrary semirings for real numbers may be used

\[
\text{FSM } M = (\Sigma_1, \Sigma_2, Q, i, c_i, F, C, E)
\]

\[
\begin{align*}
\Sigma_1 & \text{ - is a finite input alphabet} \\
\Sigma_2 & \text{ - is a finite output alphabet} \\
Q & \text{ - is a finite set of states} \\
i & \in Q \text{ - is the initial state} \\
c_i & \text{ - is the initial weight} \\
F & \subseteq Q \text{ - is the set of final states} \\
C : F & \rightarrow R \text{ - is the final weight function} \\
E & \subseteq Q \times \Sigma_1 \times \Sigma_2 \times R \times Q \text{ - is the set of transitions}
\end{align*}
\]
DFKI Finite-State Tools Overview

- efficiency-oriented implementation
- architecture and functionality is mainly based on the tools developed by AT&T
- letter transducers, no final emissions
- representation: textual format vs. compressed binary format

```
0  1.0
-----------------------
0  1  a  b  1.0
0  2  a  c  2.0
-----------------------
1  3.0
2  4.0
```

- most of the provided operations are based on recent approaches (Mohri, Pereira, Roche, Schabes)
DFKI Finite-State Tools Overview

- operations are divided into four main pools:

  converting operations:
  - converting textual representation into binary format and vice versa
  - creating graph representation for FSMs, etc.

  rational and combination operations:
  - union
  - concatenation
  - closure
  - local extension
  - reversion
  - intersection
  - inversion
  - composition

  equivalence transformations:
  - determinization
  - bunch of minimization algorithms
  - epsilon removal
  - trimming

  other:
  - extending input/output alphabet
  - determinicity test
  - collecting arcs with identical labels
  - information about an FSM
DFKI Finite-State Tools Examples

- Keywords Recognition - Automata Search

KEYWORD RECOGNITION (1)
DFKI Finite-State Tools Examples

• Keywords Recognition - Deterministic Search
DFKI Finite-State Tools Examples

- Keywords Recognition - Minimal Deterministic Search

KEYWORD RECOGNITION (3)
DFKI Finite-State Tools Examples

• contextual PART-of-SPEECH filtering rule:

if the previous word form is a **determiner** and the next word form is a **noun** then filter out the **verb reading**

example: ... die bekannten Bilder .... ("the known pictures\“)
DFKI Finite-State Tools Examples

- transform FST into an deterministic FST that operates globally on the input in one pass (local extension, trimming, determinization)
DFKI Finite-State Tools Overview

• some of the operations restricted to subclass of FSMs due to limited closure properties of FST’s (e.g., determinization, removing epsilon transitions)

• some new algorithms: local extension for WFST and direct incremental construction of minimal deterministic acyclic FSA

• improvement of existing algorithms (e.g., epsilon-removal)

• availability: user-program level vs. C++ library level (both UNIX and MS Windows)

user-program level:

```plaintext
fsm intersect in1.fsm in2.fsm out.fsm
```

C++ library level:

```plaintext
FSM in[2];
FSM out;
in[0].initialize("in1.fsm");
in[1].initialize("in2.fsm");
out = fsm_intersection(in,2);
out.save("out.fsm");
```

• simple regular compiler
# DFKI Finite-State Tools Overview

<table>
<thead>
<tr>
<th>Operation</th>
<th>AT&amp;T</th>
<th>FSA6</th>
<th>DFKI-II</th>
<th>DFK-I</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNION</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>INTERSECTION</td>
<td>+</td>
<td>+</td>
<td>FSA</td>
<td>+</td>
</tr>
<tr>
<td>CONCATENATION</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>CLOSURE</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>REVERSION</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>COMPLEMENT</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>DIFFERENCE</td>
<td>FSA</td>
<td>FSA</td>
<td>FSA</td>
<td></td>
</tr>
<tr>
<td>INVERSION</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>LOCAL EXTENSION</td>
<td></td>
<td></td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>COMPOSITION</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>EPSILON REMOVAL</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>DETERMINIZATION</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>MINIMIZATION</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>EQUALITY</td>
<td></td>
<td>FSA</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>TRIMMING</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>BEST PATHS</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PERFECT HASHING</td>
<td></td>
<td></td>
<td></td>
<td>+</td>
</tr>
<tr>
<td>VISUALIZING</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>CFG-APPROXIMATION TO FSA</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REGULAR COMPILER</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>(+)</td>
</tr>
</tbody>
</table>
DFKI Finite-State Tools Overview

- Generalized Finite-state Machine Toolbox (GFSMT) by Oliver S. Scherf

  ➢ LISP implementation
  ➢ final emissions
  ➢ transitions labeled with sequences of symbols
  ➢ arbitrary semirings
  ➢ transition table not compressed
  ➢ subtleties in implementation of algorithms