

# Sphinx-4: A Flexible Open Source Framework for Speech Recognition

Norman Rosner

Institut für Sprachwissenschaft  
Universität Potsdam

03. Juni 2008

# Outline

- 1** Anforderungen an Sphinx
- 2 Anforderungen und Aufbau
- 3 Module

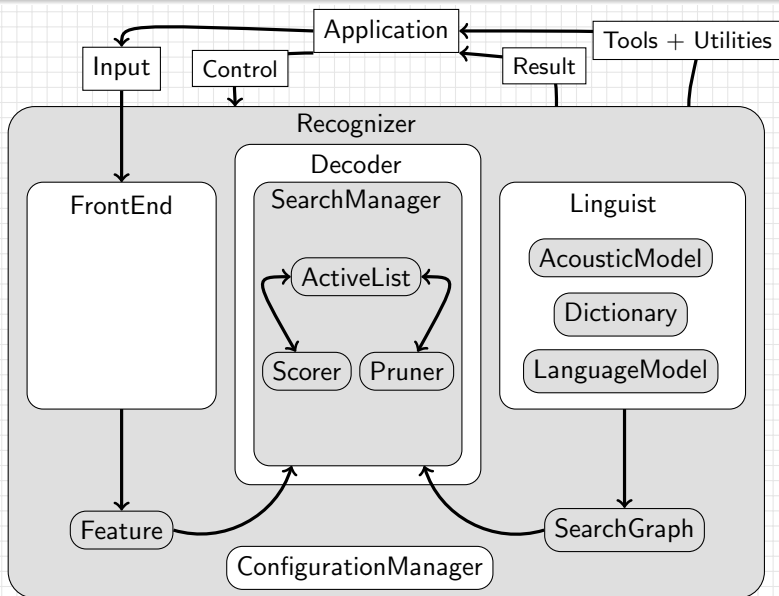
# Flexibel, Open Source und was nicht alles

- sowohl in der Forschung als auch industriell einsetzbar
- einsetzbar auf einer Vielzahl von Plattformen und Betriebssystemen
- umfassende “state of the art design-patterns” existierender Systeme
- aber gleichzeitig Unterstützung für neue, sich entwickelnde Technologien
- modular: separierte Komponenten für einzelne Aufgaben
- steckbar/ansteckbar (pluggable): Module während der Laufzeit austauschbar

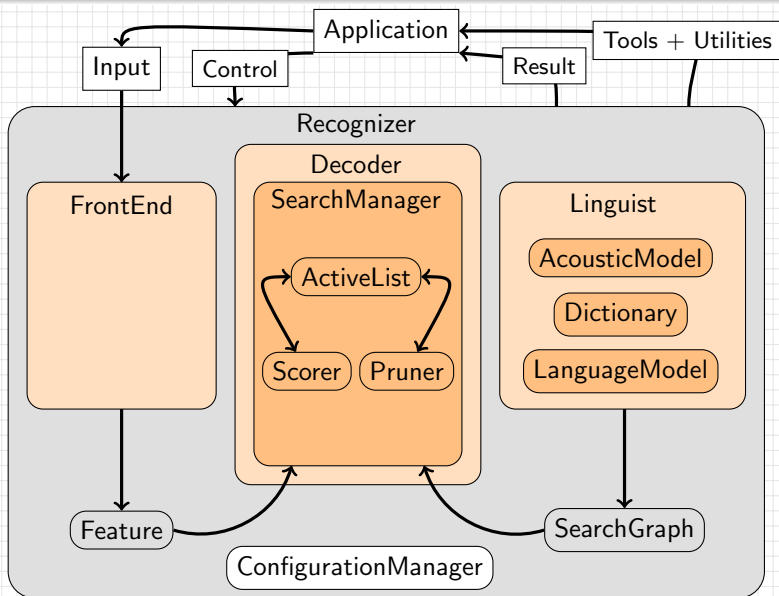
# Outline

- 1 Anforderungen an Sphinx
- 2 Anforderungen und Aufbau
- 3 Module

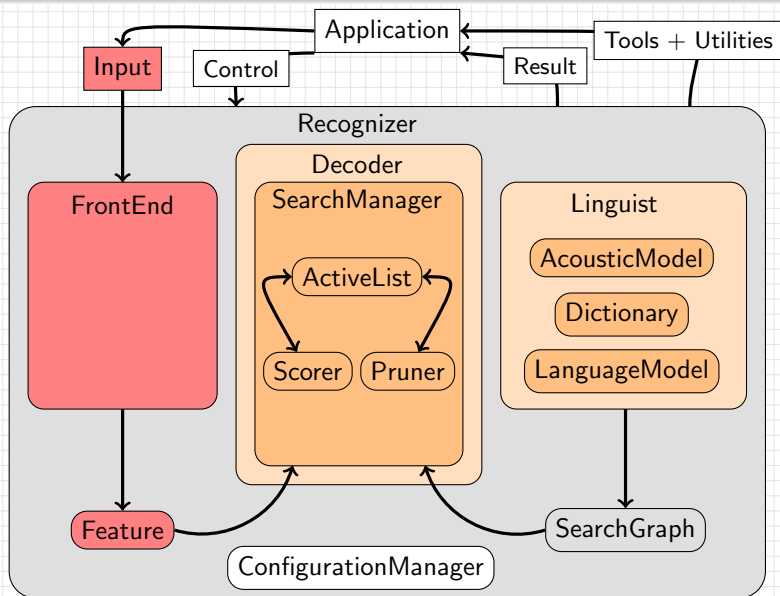
## Aufbau



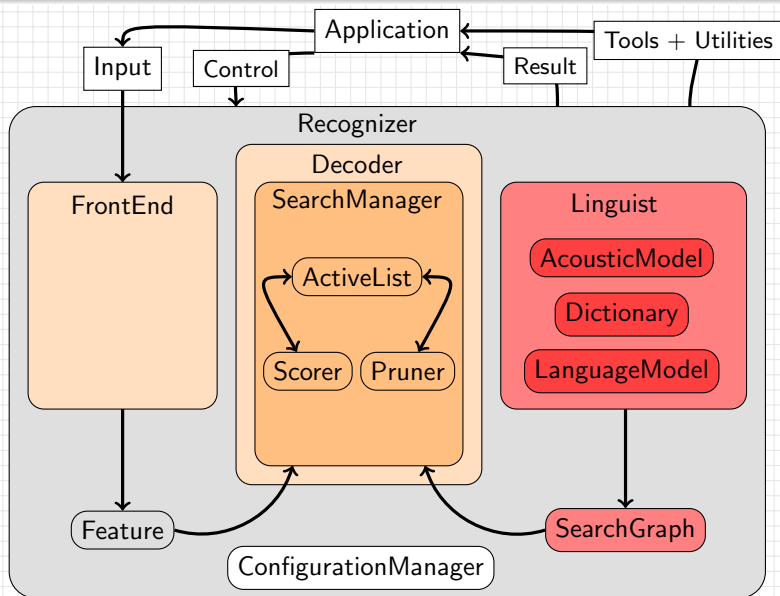
## Aufbau



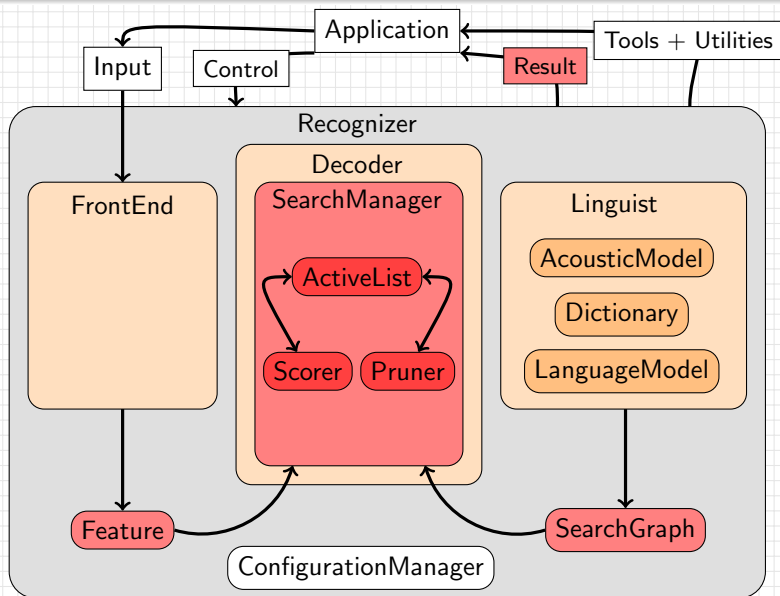
# Aufbau



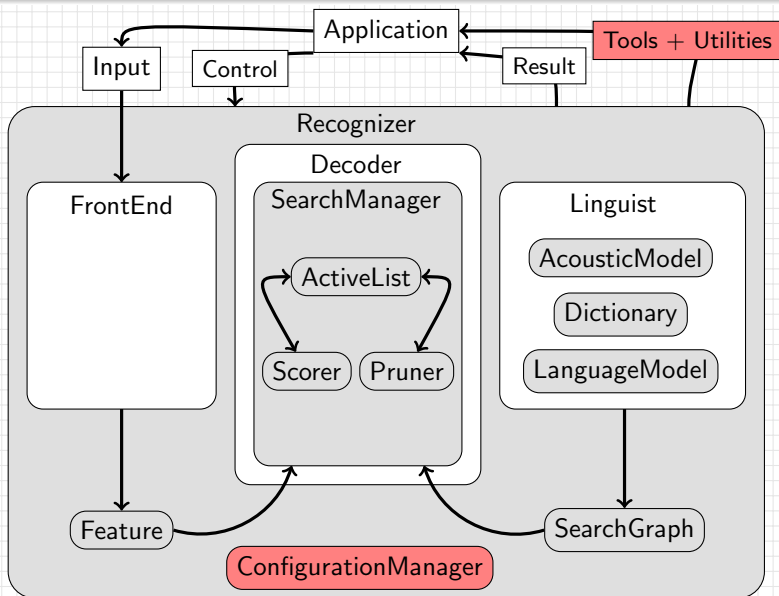
# Aufbau



# Aufbau



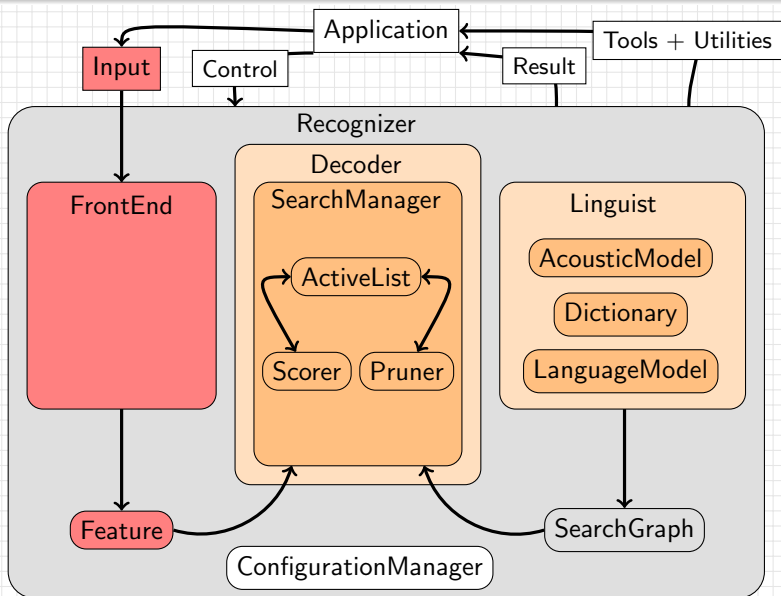
## Aufbau



# Outline

- 1 Anforderungen an Sphinx
- 2 Anforderungen und Aufbau
- 3 Module**

# Übersicht

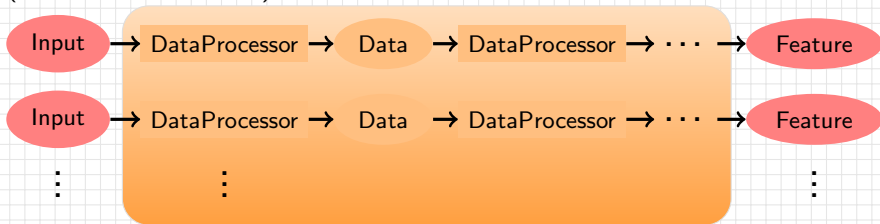


# Übersicht

- parametrisiert ein Eingangssignal (*Input*)
- erzeugt Merkmale (*Features*)
- serielle (miteinander kommunizierende) signalverarbeitende Module (*DataProcessors*, DP)

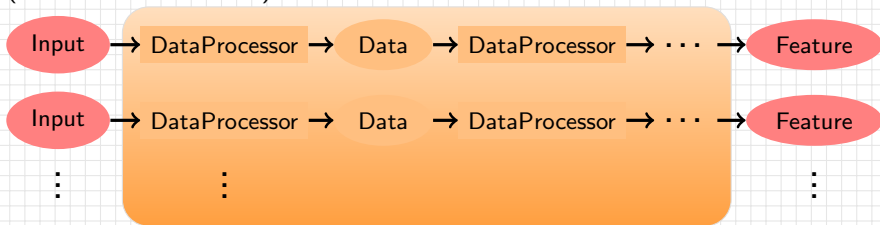
# Übersicht

- parametrisiert ein Eingangssignal (*Input*)
- erzeugt Merkmale (*Features*)
- serielle (miteinander kommunizierende) signalverarbeitende Module (*DataProcessors*, DP)



# Übersicht

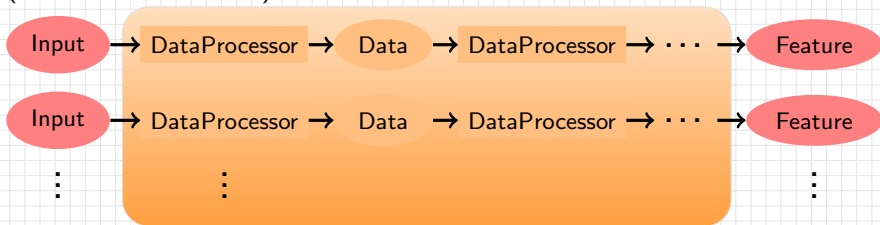
- parametrisiert ein Eingangssignal (*Input*)
- erzeugt Merkmale (*Features*)
- serielle (miteinander kommunizierende) signalverarbeitende Module (*DataProcessors*, DP)



- beliebig lange Verarbeitungsketten: Eingabe und Ausgabe eines jeden DataProcessors sind generische Daten-Objekte mit verarbeiteten Eingabedaten
- beliebige Anzahl von Verarbeitungsketten

# Übersicht

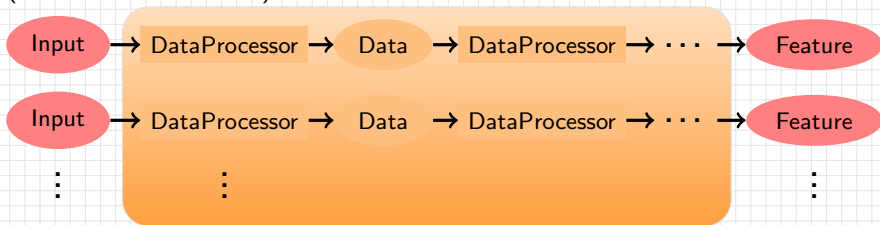
- parametrisiert ein Eingangssignal (*Input*)
- erzeugt Merkmale (*Features*)
- serielle (miteinander kommunizierende) signalverarbeitende Module (*DataProcessors*, DP)



- Kommunikation zwischen Ketten: Pull-Design
  - Ergebnisse eines DPs nur wenn es vom anderen benötigt wird
  - DPs funktionieren als Speicher/Buffer
  - verschiedene Suchverfahren (im *Decoder*) möglich

# Übersicht

- parametrisiert ein Eingangssignal (*Input*)
- erzeugt Merkmale (*Features*)
- serielle (miteinander kommunizierende) signalverarbeitende Module (*DataProcessors*, DP)

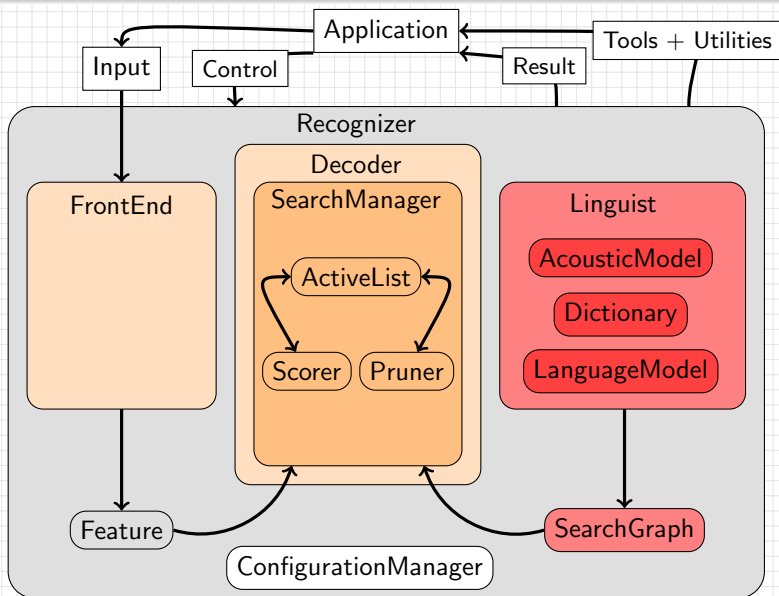


- gleichzeitige Dekodierung durch verschiedene Parametertypen
- damit auch Parameter von nichtsprachlichen Signalen (z.B. Video) möglich

# Übersicht

- parametrisiert ein Eingangssignal (*Input*)
- erzeugt Merkmale (*Features*)
- serielle (miteinander kommunizierende) signalverarbeitende Module (*DataProcessors*, DP)
- generische Architektur des Moduls! ermöglicht verschiedenste Vorverarbeitung, z.B.:
  - Einlesen aus einer Vielzahl von Formaten (Batch-Modus)
  - system audio input device (Mikrofon)
  - Vorverstärkung
  - **F**ast **F**ourier **T**ransformation um Grundfrequenzen zu berechnen
  - und und und

# Übersicht



# Übersicht

- generiert den Suchgraphen (*SearchGraph*), der vom *Decoder* benötigt wird
- basiert auf der Sprachstruktur, gegeben durch das *LanguageModel*, der topologischen Struktur des akustischen Modells (*AcousticModel*) und einem Wörterbuch (*Dictionary*)
- alle der drei Komponenten sind selbstverständlich steckbar (pluggable)

# LanguageModel

- repräsentiert die Struktur auf Wortebene
- verschiedenste Implementierungen
- typischerweise zwei Typen von Implementierungen:
  - 1 graph-getriebene Grammatiken
  - 2 stochastische N-Gram-Modelle
- einige Beispiele an implementierten Grammatiken:  
*SimpleWordListGrammar, JSGFGrammar, LMGrammar, FSTGrammar, SimpleNGramModel*

# Dictionary

- Zerlegung der Wörter aus dem LanguageModel in Sequenzen aus dem AcousticModel
- verschiedene Implementierungen

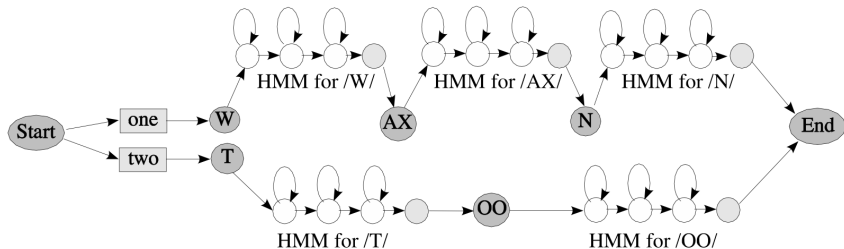
# AcousticModel

- Abbildung von sprachlichen Einheiten auf HMMs (Hidden Markov Models)
- Kontextinformationen und Wortposition kann zur Bewertung gegen die Features mit herangezogen werden
- geschickte Implementierung der HMMs:
  - HMM = gerichteter Graph von Objekten
  - Zugriff auf Informationen anderer Zustände in Zuständen
  - Verteilung von Informationen dadurch sehr feingliedrig

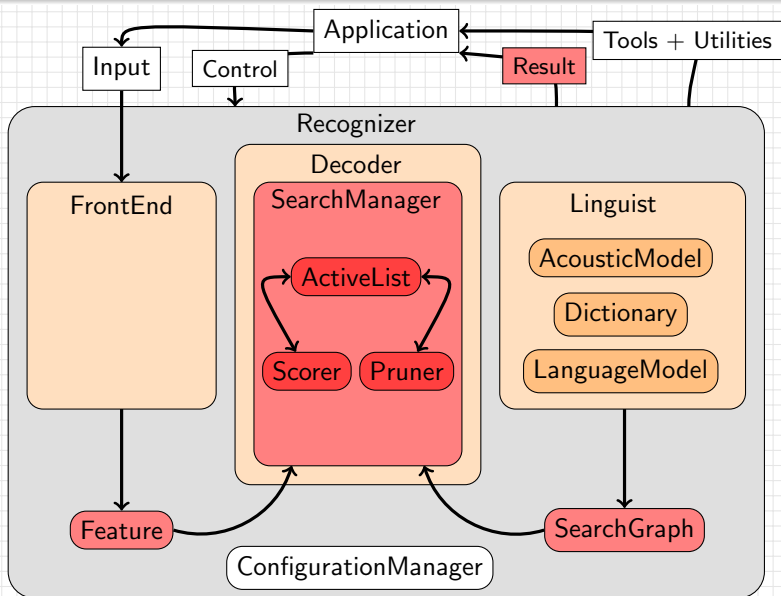
# SearchGraph

- repräsentiert den Suchraum
- gerichteter Graph mit emittierenden und nichtemittierenden Zuständen
- emittierende Zustände können gegen Merkmale gewertet werden
- nichtemittierender Zustand = höhere linguistische Konstrukte (Wörter, Phoneme)
- generische Implementierung in Bezug auf:
  - Struktur des Suchraumes
  - Größe des phonetischen Kontextes
  - Grammatiktyp
  - Verhalten des Zustandes
- tatsächliche Implementierung beeinflusst Speicherauslastung, Geschwindigkeit, Genauigkeit der Erkennung
- modularer Aufbau von Sphinx: verschiedene Strategien des Graphen während der Laufzeit

## SearchGraph



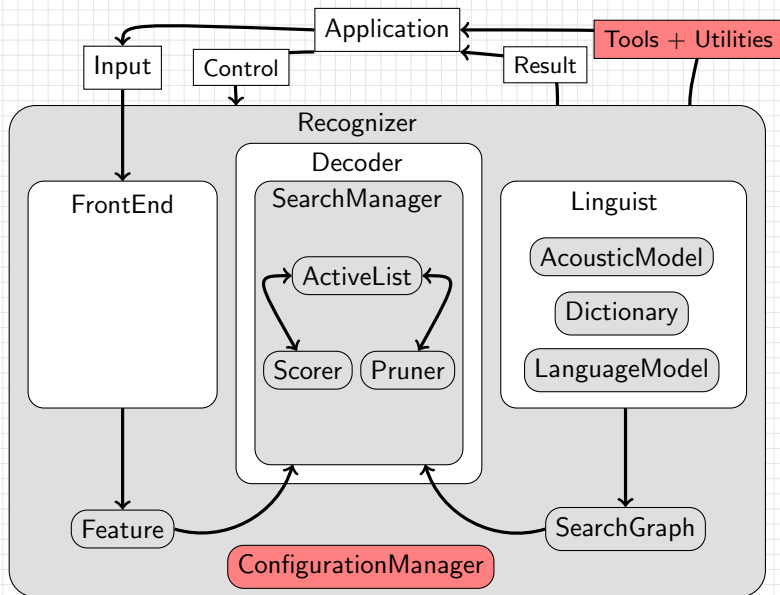
## Decoder



# Decoder

- Generierung von Resultaten durch Zusammenführung von Merkmalen und Suchgraph
- durch die Resultate ist Unterstützung durch die Anwendung möglich
- auch hier: generische Implementierung
- verschiedenste Suchalgorithmen denkbar (A\*, Viterbi, bi-direktional ...)
- Sub-Framework bestehend aus *ActiveList*, *Scorer* und *Pruner*
- *ActiveList*: beinhaltet aktive Token aus dem Suchraum (geschieht mithilfe des *Pruners*)
- *Scorer*: gewichtet Merkmale wenn vom *SearchManager* benötigt (Modularität erneut von Vorteil)

## Konfiguration



# Konfiguration

- Zusammenstecken der Module geschieht über den Konfigurationsmanager
- Java-Objekt
- Konfigurationen sind entweder in XML-Dateien angegeben oder *on-the-fly* änderbar
- Rechnerdaten sind mithilfe von *Tools + Utilies* überwachbar

# Outline

4 Anhang

5 Literatur

# Beispiel: SimpleWordListGrammar

zero  
one  
two  
three  
four  
five  
six  
seven  
eight  
nine

## Beispiel: FSTGrammar

```
I 2
F 0 2.30259
T 0 1 <unknown> <unknown> 2.30259
T 0 4 wood wood 1.60951
T 0 5 cindy cindy 1.60951
T 0 6 pittsburgh pittsburgh 1.60951
T 0 7 jean jean 1.60951
F 1 2.89031
T 1 0 , , 0.587725
T 1 4 wood wood 0.58785
F 2 3.00808
T 2 0 , , 0.705491
T 2 1 <unknown> <unknown> 0.58785
F 3 2.30259
T 3 0
F 4 2.89031
T 4 0 , , 0.587725
T 4 6 pittsburgh pittsburgh 0.58785
F 5 2.89031
T 5 0 , , 0.587725
```

# Outline

4 Anhang

5 Literatur

Willie Walker, Paul Lamere, Philip Kwok, Bhiksha Raj, Rita Singh, Evandro Gouvea, Peter Wolf, and Joe Woefel. Sphinx-4: A Flexible Open Source Framework for Speech Recognition. Technical report, Sun Microsystems, Carnegie Mellon University and Mitsubishi Electric Research Labs, 2004.