# Ein Satz von Komponenten zur nachhaltigen Verwaltung und Umwandlung von frei formatierten Wörterbuchdaten

Prof. Dr. Klaus Lagally
Universität Stuttgart
Institut für Informatik
Breitwiesenstraße 20-22
D-70565 Stuttgart *

PD Dr. Lutz Rzehak
Humboldt-Universität zu Berlin
Zentralasien-Seminar
Unter den Linden 6
D-10099 Berlin †

27. August 2002

**Zusammenfassung**

Wir schlagen vor, bei der Verarbeitung linguistischer Informationen die Darstellung der Daten anwendungsunabhängig zu halten und die spätere Verarbeitung davon strikt zu trennen. Wir erreichen damit nicht nur eine strenge Arbeitsteilung, sondern auch flexible Wiederverwendbarkeit. Wir demonstrieren das Konzept an einem einfachen, aber repräsentativen Prototyp eines mehrsprachigen Wörterbuches in unterschiedlichen externen Darstellungen, die aus derselben Wissensbasis gewonnen werden. Dabei zeigt sich, daß für die Ablage der Daten sowohl eine Codierung im TEX-Format wie auch eine XML-Darstellung gleichermaßen in Frage kommen, und daß sich die weiteren Auswertungen und Umwandlungen, zumindest wenn man TEX als universelle Plattform verwendet, in sehr komfortabler Weise kombinieren lassen.

*E-Mail: `mailto:lagally@informatik.uni-stuttgart.de`
WWW: `http://www.informatik.uni-stuttgart.de/ifi/bs/people/lagally.htm`
†E-Mail: `mailto:lutz.rzehak@rz.hu-berlin.de`
WWW: `http://www2.hu-berlin.de/zentralasien/`

# 1 Motivation

Äußerer Anlaß der vorliegenden Untersuchung war ein eher zufälliger Kontakt der beiden Autoren, wobei es um recht technische Fragen der graphischen Aufbereitung von Texten in einer erweiterten arabischen Schrift ging. Der erste Autor (K.L.) hatte sich seit etwa 10 Jahren mit der Darstellung der (auch erweiterten) arabischen Schrift auf verbreiteten Computersystemen beschäftigt, und das dabei entstandene System ArabTEX[1] bietet sich zur Verwendung im Projekt des zweiten Autors (L.R.) an. Dabei geht es darum, eine umfangreiche Sammlung von Wörtern und Wortformen aus der in Belutschistan gesprochenen Sprache samt ihren Entsprechungen in Paschto und Persisch sowie einer Übersetzung in europäische Sprachen (hier i.d.Regel Englisch) in Gestalt eines Wörterbuchs zugänglich zu machen. Der dabei nötige Aufwand zur graphischen Aufbereitung ist sehr erheblich, aber schematisch und inhaltlich wenig anspruchsvoll, sobald die Ausgangsdaten einmal vollständig erfaßt sind, und dieser Schritt sollte daher soweit möglich automatisiert werden. Auch diese Aufgabe ist nicht trivial, hat aber mit Linguistik und Philologie wenig zu tun, und so bietet sich eine interdisziplinäre Zusammenarbeit zwischen einem Philologen und einem Informatiker geradezu an.

Wenn ein Philologe und ein Informatiker in einem solchen Projekt fruchtbar zusammenarbeiten sollen, ohne daß jeder vom Fachgebiet des anderen allzuviel verstehen muß, so spielt die Darstellung der Daten eine entscheidende Rolle. Sie muß sowohl maschinenlesbar wie auch für menschliche Leser leicht handhabbar sein, und sie sollte keine spezielle Ausrüstung in Hardware und Software erfordern, vielmehr nach Möglichkeit auf allen Computersystemen in gleicher Weise verarbeitet werden können. Der Philologe sollte in der Lage sein, den Datenbestand weiterhin zu pflegen, ohne sich um Einzelheiten der graphischen Aufbereitung zu kümmern; und der Informatiker sollte diese Aufbereitung ohne tiefergehende linguistische Kenntnisse leisten können. Das bedeutet, daß jegliche linguistische Information über die Daten im Datenbestand in einer Weise eingeschlüsselt wird, daß sie leicht automatisch wiedergewonnen werden kann, aber auch nicht mehr: Aspekte der späteren Aufbereitung spielen hier keine Rolle. Damit bekommt der Datenbestand Archivcharakter: das Wissen ist längerfristig festgehalten und kann später auch nach anderen Gesichtspunkten ausgewertet werden, als die ursprüngliche Anwendung nahelegt, und an die früher gar nicht gedacht war.

Wir untersuchen im folgenden die logische Struktur unseres Datenbestandes und mögliche technische Realisierungen. Vollständigkeit ist dabei nicht unser Ziel, vielmehr genügt uns die Realisierung eines Prototyps.

[1] `ftp://ftp.informatik.uni-stuttgart.de/pub/arabtex/arabtex.htm`

## 2 Abstrakte Syntax

Wir versuchen im folgenden, die logische Struktur unserer Wissensbasis so präzise festzulegen, daß eine automatische Analyse möglich ist, ohne aber dabei die weitere Verarbeitung oder auch nur die konkrete Ablage zu präjudizieren. In der Sprechweise der Informatik handelt es sich um die „abstrakte Syntax“; Linguisten mögen uns diesen laxen Sprachgebrauch nachsehen.

Unser gedachtes *Wörterbuch* besteht aus einer *Menge* von *Einträgen*, deren jeder einem *Lemma* zugeordnet ist. Unter einem Lemma verstehen wir dabei ein Lexem in der Sprache, auf die sich das Wörterbuch bezieht (hier: Belutschi) oder ein Wortgruppenlexem, das aus zwei oder mehr lexikalischen Einheiten besteht, die zusammen einen Lexikoneintrag bilden. Homonyme und Allographen unterscheiden wir, wenn wir sie auseinanderhalten wollen, durch *Zusatzangaben* (etwa willkürlich vergebene laufende Nummern), betrachten sie also dann als verschiedene Lemmata. Damit gehört zu jedem aufgenommenen Lemma ein einziger Eintrag, der jeweils das dazu festgehaltene Zusatzwissen enthält. Die Menge der Lemmata ist *logisch* ungeordnet; in einer konkreten Darstellung liegen die Elemente natürlich in irgend einer Reihenfolge vor, und sie lassen sich bei Bedarf auch nach geeigneten Kriterien sortieren und durchsuchen.

Ein Eintrag ist entweder ein *Querverweis* von einer Form aus dem Paradigma eines Lexems, die einen eigenständigen Eintrag als Lemma rechtfertigt, zur entsprechenden Grundform als Basiseinheit des Lexikons, also ein Querverweis auf einen Eintrag für ein anderes Lemma, oder er gibt zu einem Lemma zusätzliches Wissen in Form von *Attributen* an. Das können Angaben zur *Umschrift* oder zur grammatikalischen *Funktion* sein, oder auch *Übersetzungen* des Lemmas in andere Sprachen; dabei muß natürlich jeweils die betreffende *Sprache* angegeben werden. Unsere Darstellung ist grundsätzlich offen für Erweiterungen. Auch weitere Attribute, an die wir hier noch nicht denken, sollen später zugefügt werden können; spätestens zur Zeit der Verarbeitung muß dann allerdings erkennbar sein, welcher Art das Attribut ist, und dann erst muß festgelegt werden, wie es zu behandeln ist.

Manchmal wechselt ein Lexem im Kontext die Bedeutung, oder es geht stehende Verbindungen ein, die speziell zu interpretieren und zu übersetzen sind. *Beispiele* dazu gehören logisch zum Lemma, und wir ordnen sie jeweils dessen Eintrag zu; auch sie können wiederum Attribute tragen.

Damit haben wir den logischen Aufbau unserer Daten im wesentlichen schon festgelegt. Etwas formeller können wir unser Ergebnis durch ein Fragment einer kontextfreien Grammatik ausdrücken; dabei haben wir uns um „sprechende“ Bezeichner bemüht und lassen hier einige Termini noch offen, deren Bedeutung klar sein sollte. Ein Vorteil der formellen Festlegung ist die Möglichkeit einer automatischen Konsistenzüberprüfung, zumindest des Eingabeformates.

Wörterbuch = Eintrag*
Eintrag = Lemma Verweis
Eintrag = Lemma Attribut*
Lemma = Text Variante*
Verweis = Lemma
Eintrag = Eintrag Beispiel
Attribut = Transkription
Attribut = Grammatikangabe
Attribut = Übersetzung
Übersetzung = Sprache Text
Beispiel = Text Attribut*

Die hier verwendete EBNF-Notation ist dem Informatiker geläufig und für Philologen zumindest leicht zu erraten. Sie läßt sich, wenn man will, kompakter schreiben:

Wörterbuch = Eintrag*
Eintrag = Lemma ( Verweis | Attribut* ) Beispiel*
Lemma = Text Variante*
Verweis = Lemma
Attribut = Transkription | Grammatikangabe | Übersetzung
Übersetzung = Sprache Text
Beispiel = Text Attribut*

# 3 Konkrete Codierungen

Bei der Festlegung einer konkreten Darstellung unseres Datenmaterials können wir uns von Zweckmäßigkeitsüberlegungen leiten lassen. Die Darstellung muß erlauben, die in der abstrakten Syntax beschriebene logische Gliederung wiederzugeben, soll aber noch keine bestimmte Art der späteren Auswertung soweit präjudizieren, daß alternative Verarbeitungen ausgeschlossen oder zumindest stark erschwert werden. Weiterhin soll sie weder spezielle Erfassungs- und Darstellungs-Programme noch die Wahl bestimmter Computermodelle voraussetzen.

Damit bietet sich eine Darstellung als reine Textdatei an, wie sie auf allen uns bekannten Computermodellen und in allen verbreiteten Betriebssystemen mit einfachsten Texteditoren erzeugt und bearbeitet werden kann; die

Verwendung etwa vorhandener komfortabler Erfassungs- und Bearbeitungsprogramme wird dadurch nicht ausgeschlossen. Als Zeichensatz verwenden wir den internationalen Minimal-Standard: 7-Bit-ASCII-Codierung[2] ohne nationale Besonderheiten; damit sind die Daten problemlos bei Bedarf auch über Computernetze und auf andere Plattformen ohne Umcodierungen übertragbar. Eine bestimmte Zeilenstruktur der Einträge schreiben wir nicht vor, halten diese vielmehr formatfrei und selbstbeschreibend; dies mag die manuelle Erfassung erleichtern. Nur aus Bequemlichkeitsgründen setzen wir die Einträge gelegentlich durch Leerzeilen voneinander ab.

Für die Festlegung des Aufbaues der Einträge haben wir nun noch mehrere Möglichkeiten, die von ihrer Ausdruckskraft her gleich mächtig sind, aber sich im Aufwand für die Auswertung sehr stark unterscheiden können. Solange jede Darstellung die gesamte Information enthält, lassen sich die anderen grundsätzlich bei Bedarf daraus automatisch gewinnen.

Für Archivzwecke könnte eine Darstellung nach XML-Konventionen[3] grundsätzlich interessant sein, doch existiert für unsere erste beabsichtigte Auswertung, nämlich den Aufbau eines Wörterbuchs in gedruckter Form, bislang kein sich auf XML stützendes Auswertungsprogramm, sondern ein solches müßte erst entwickelt werden. Die einzige derzeit existierende Möglichkeit, die Textanteile in erweiterter arabischer Schrift darzustellen, ist die Verwendung des Systems ArabTEX[4], das sich wiederum auf das Satzsystem TEX von D.E.Knuth[5] abstützt; es müßte also als Zwischenform eine Darstellung erzeugt werden, die von TEX und ArabTEX verarbeitet werden kann.

Eine solche Zwischenform kann nun so gestaltet werden, daß sie sich nicht nur als Eingabe zum Textsatz eignet, sondern auch noch alle Information nach der abstrakten Syntax enthält, sodaß sie auch für andere Auswertungen und Umwandlungen als Eingabe brauchbar ist. Das liegt daran, daß TEX nicht nur ein Satzsystem ist, sondern auch auf sehr flexible (wenn auch gewöhnungsbedürftige) Weise für beliebige Verarbeitungen programmiert werden kann. Wir legen daher hier eine Darstellung der Einträge nach TEX-Konventionen fest; für die fremdsprachlichen Angaben verwenden wir die von ArabTEX verarbeitete Eingabenotation.

---

[2]International Organization for Standardization: *Information processing – ISO 7-bit coded character set for information interchange.* ISO 646

[3]`http://www.w3.org/XML/`

[4]`ftp://ftp.informatik.uni-stuttgart.de/pub/arabtex/arabtex.htm`

[5]`http://www-cs-faculty.stanford.edu/~knuth/`

## 3.1 TeX-Codierung

TeX unterstützt eine Makrosprache, die unter anderem Kommandos mit Argumenten enthält; die Auswirkung eines Kommandos bei der Ausführung kann frei definiert werden im Rahmen der verfügbaren Grundfunktionen. Wir bilden nun formal jeden Eintrag auf ein neues solches Kommando ab, und dasselbe gilt auch für seine Bestandteile, die jeweils ein Attribut und dessen Wert bezeichnen. Die Bedeutung der Kommandos lassen wir bewußt offen und behalten uns vor, sie erst bei der Verarbeitung festzulegen; erst dann bewirkt die Ausführung der Kommandos die gewünschte Auswertung. Die Kommandonamen sind willkürlich, sie dürfen nur nicht mit bereits vorhandenen Kommandos kollidieren. Als Beispiel geben wir eine mögliche konkrete Ausprägung unserer abstrakten Syntax an:

```
Wörterbuch = "\Dictionary{" Text "}" Eintrag*
Eintrag = Lemma Verweis
Eintrag = Lemma Attribut*
Lemma = "\Lemma{" Text "}" Variante*
Variante = "\Variante{" Zahl "}"
Verweis = "\Verweis{" Text "}"
Eintrag = Eintrag Beispiel
Attribut = "\Transkription{" Text "}"
Attribut = "\Grammatikangabe{" Text "}"
Attribut = Übersetzung
Übersetzung = Sprache "{" Text "}"
Sprache = "\Belutschi"
Sprache = "\Paschto"
Sprache = "\Dari"
Sprache = "\Englisch"
Beispiel = "\Beispiel{" Text "}" Attribut*
```

oder mit kürzeren Kommandos, um bei der Erfassung Mühe zu sparen:

```
W\"orterbuch = "\Dict{" Text "}" Eintrag*
Lemma = "\Lm{" Text "}" Variante*
Variante = "\Va{" Zahl "}"
Verweis = "\cf{" Text "}"
Attribut = "\Tr{" Text "}"
Attribut = "\Gr{" Text "}"
Attribut = Sprache "{" Text "}"
Sprache = "\Be" | "\Pa" | "\Da" | "\En"
Beispiel = "\Ex{" Text "}" Attribut*
```

Der hier nicht erklärte Parameter „Text“ wird erst zum Zeitpunkt der Auswertung von einem geeignet definierten Kommando kontextabhängig interpretiert; Fehler dabei sind inhaltlicher Art und betreffen den Philologen.

Zur Auswertung ist nun ein *Hauptdokument* zu erstellen, das die allgemeine Formatierung definiert, und eine *Stildatei* mit den Definitionen der Kommandos einliest; Einlesen und Ausführen der Daten *als Kommandofolge* erledigt den Rest. Wir geben im folgenden einige Beispiele; der vorliegende Bericht ist dabei das gemeinsame Hauptdokument.

## 3.2 Sequentielle Verarbeitung

Als gemeinsame Grundlage für die folgenden Auswertungsbeispiele verwenden wir einen kleinen, aber repräsentativen Ausschnitt aus der Wissensbasis, der alle wesentlichen Spezialfälle enthält. Am Kopf steht der lokale Name der Wissensbasis, das Datenende ist mit einem Blindeintrag markiert.

```
% rzdict1.dat
\Dict {Belutschi} % Testdaten im TeX-Format

\Lm {sA}        % Lemma in Belutschi
 \Va {1}        % Variante
 \Tr {s\=a}     % Transkription
 \Gr {n}        % Grammatikalische Kategorie
 \Pa {Orbe,sI}  % Paschto
 \Da {^gaw}     % Dari
 \En {barley}   % Englisch

\Lm {sA}        % Lemma
 \Va {2}        % Variante
 \cf {sAtin}    % Querverweis

\Lm {sAp}
 \Tr {s\=ap}
 \Gr {.s}
 \Da {.sAf}
 \En {clean, pure}

\Lm {sApdil}
 \Tr {s\=apdil}
 \Gr {.s}
 \Pa {spInY, spIn ze,rY}
```

```
 \Da {.sAf dil}
 \En {honest, sincere, frank}

\Lm {sAt}
 \Tr {s\=at}
 \Gr {n}
 \Pa {gA.nah, zEwar}
 \Da {zEwar, ^gawAhirAt}
 \En {jewellery; precious stones; gems}

\Lm {sAtin, sAtag}
 \Tr {s\=atin, s\=atag}
 \Gr {fm}
 \Pa {sAtil}
 \Da {nigahdA^stan, nigAh kardan}
 \En {protect, defend, take care of, guard}
\Ex {man AyirA sAtIn}      % Beispiel
 \Pa {zeh ha.geh sAtem}
 \Da {man UrA nigAh mEkunam}
 \En {I protect him}
\Ex {ma.hmUd AyirA sAtIt}
 \Pa {ma.hmUd ha.geh sAtI}
 \Da {ma.hmUd UrA nigAh mEkunad}
 \En {Mahmud protects him}
\Ex {ma.hmUd AyirA sAtit}
 \Pa {ma.hmUd ha.geh wusAteh}
 \Da {ma.hmUd UrA nigAh kard}
 \En {Mahmud protected him}

\Lm {sAtOk}
 \Tr {s\=at\=ok}
 \Gr {fA}
 \Pa {sAtUnkY}
 \Da {nigahbAn}
 \En {protector; defender; guard, watchman}

\Lm {*} % Datenende
```

Bei einem wirklich großen Wörterbuch wollen wir nicht annehmen, daß die gesamten Daten gleichzeitig im Speicher liegen müssen; vielmehr verarbeiten wir die Einträge einzeln sequentiell nacheinander. Dazu brauchen wir ein Treiberprogramm, das beim Einlesen der Daten durch deren erstes Kommando gestartet wird und für jeden Eintrag eine Verarbeitungsroutine anstößt, die noch über eine Stildatei frei vorgegeben werden kann, ebenso wie eine Anfangs- und Endebehandlung. Der Aufbau des Treiberprogramms hängt nur davon ab, wie das erste Kommando heißt, und wie die Einträge getrennt und beendet werden.

```
% rzdict3b.tex
% Treiber fuer Woerterbuch als \Dict{Titel} Eintrag* Datenende
% jeder Eintrag beginnt mit \Lm
% Datenende: \Lm{*}
% \Process verarbeitet den Eintrag

\long\def \Dict #1#2\Lm {% Eintraege verarbeiten
\Kopf {#1}\Eintrag \Lm }

\long\def \Eintrag \Lm #1#2\Lm #3{%     ganzen Eintrag einlesen
\Process {\Lm{#1}#2}%    Rumpf ausfuehren, evtl. weitermachen
\cond{\ifx *#3}{\Fuss }{\Eintrag \Lm {#3}}\fi }
```

Im Interesse späterer Anwendungen haben wir die vorkommenden Kommandos auch bei ArabTEX angemeldet.

```
\allowarab \Dict \allowarab \Kopf \allowarab \Eintrag
\allowarab \Process \allowarab \Fuss
```

## 3.3 Syntaxüberprüfung

Bei der Erfassung der Rohdaten im vorgeschlagenen Format kommt es mit Sicherheit gelegentlich zu Fehlern inhaltlicher und auch formaler Art. Inhaltliche Fehler zu beseitigen, erfordert philologische Kenntnisse und setzt zumindest die Möglichkeit zu einer rudimentären Auswertung und Darstellung voraus; diese kann aber durch formale Fehler gestört oder gar verhindert werden, daher sollte zumindest die formale Einhaltung der Syntax automatisch überprüft werden können. Dies kann aber bereits der Fehlerbehandlungsmechanismus von TEX leisten; formale Fehler führen zu einer Fehlernachricht oder zumindest zu einer (womöglich schlimm formatierten) Ausgabe, an Hand deren der Fehler lokalisiert werden kann.

Dazu verwenden wir eine spezielle Stildatei, die bei formal korrektem Aufbau der Daten überhaupt keine Ausgabe liefert.

```
% rzdict2b.tex
% Test auf Syntaxfehler

\def \Kopf #1{% Beginn der Verarbeitung
  \par Syntaxtest von (#1)\par }

\long\def \Process #1{% Rumpf verarbeiten, hier: ausfuehren
  #1\relax }

\def \Fuss {% Ende der Verarbeitung
  \par Ende Syntaxtest \par }

\newcommand \Lm [1]{}
\newcommand \Va [1]{}
\newcommand \cf [1]{}
\newcommand \Tr [1]{}
\newcommand \Gr [1]{}
\newcommand \Be [1]{}
\newcommand \Pa [1]{}
\newcommand \Da [1]{}
\newcommand \En [1]{}
\newcommand \Ex [1]{}
```

Diese Stildatei führt alle Felder jedes Eintrags in der gegebenen Reihenfolge je einmal aus und überprüft dabei nur die korrekte Klammerung und mögliche Namenskollisionen mit LaTeX. Die Klammerinhalte zu überprüfen erfordert menschlichen Sachverstand. Ihr Aufruf in der Form

```
\input rzdict3b.tex
\input rzdict2b.tex
\input rzdict1.dat
```

liefert weiter nichts als die Start- und Endemeldung:

Syntaxtest von (Belutschi)

Ende Syntaxtest

Bei der Vorbereitung der Daten haben wir zwei falsch stehende Klammern und einen Schreibfehler in einem Kommandonamen gefunden; diese Fehler hatten wir allerdings zum Test eigens vorher eingebaut.

# 4 Externe Darstellung im TEX-Format

Wir geben im folgenden einige typische Beispiele für mögliche Aufbereitungen der Daten, ausgehend von einer Darstellung in unserem TEX-Format; die Eingabedatei ist stets dieselbe, doch verwenden wir jeweils andere Stildateien, auch der Kontext der Auswertung ist unterschiedlich.

## 4.1 Rechtsläufiges Wörterbuch

Als erstes Beispiel zeigen wir eine mögliche Formatierung der Daten als rechtsläufige Wörterliste. Die verwendete Stildatei definiert für jedes Datenfeld das betreffende Kommando so, daß es beim Aufruf die zugehörige externe Ausgabe absetzt.

```
% rzdict4c.tex
% Ausgabe als Liste rechtslaeufig

\def \Kopf #1{% Beginn der Verarbeitung
  {\large Rechtsl"aufige W"orterliste von Datei ‘‘#1’’}\par}

\long\def \Process #1{% Rumpf verarbeiten, hier: ausfuehren
  #1\relax }

\def \Fuss {% Ende der Verarbeitung
  \par Ende W"orterliste \par \endinput }

\def \Lm #1{\par                          % Lemma
  {\Large\setnashbf\setbalut\RL{#1}}}
\def \Va #1{${}^{#1}$ }                   % Varianten-Nr
\def \cf #1{$\rightarrow$ \RL{#1}}        % Verweis
\def \Tr #1{\emph{#1}}                    % Transkription
\def \Gr #1{\setarab\RL{#1}}              % Grammatikangaben
\def \Be #1{$|$ \setbalut\RL{#1}}         % Belutschi
\def \Pa #1{$|$ \setpashto\RL{#1}}        % Pashto
\def \Da #1{$|$ \setfarsi\RL{#1}}         % Dari
\def \En #1{#1}                           % Englisch
\def \Ex #1{\par                          % Beispiel
  {$\bullet$ \large\setnashbf\setbalut\RL{#1}}}
```

Die Stildatei ist von einer trügerischen Einfachheit, denn die Hauptarbeit der Aufbereitung wird von dem ArabTEX-Kommando \RL geleistet, und wir haben es uns auch sonst sehr einfach gemacht, indem wir nämlich die Attribute in der Reihenfolge ihres Auftretens in den Eingabedaten ausgegeben haben, was zufällig hier gerade paßt. Im allgemeinen Fall müßten wir im Kommando `\Process` den Eintrag in seine Komponenten zerlegen und deren Ausgabe dann in der von uns jeweils gewünschten Reihenfolge anstoßen.

## Rechtsläufige Wörterliste von Datei “Belutschi”

**سَا** 1 *sā* ن | أوربَشِي | جَو barley

**سَا** 2 → سَاتِن

**سَاپ** *sāp* ص | صَاف clean, pure

**سَاپدِل** *sāpdil* ص | زَرَہِی سپِین ، سپِینَی | دِل صَاف honest, sincere, frank

**سَات** *sāt* ن | زَیوَر ، گَانَه | جَوَاهِرَات ، زیوَر jewellery; precious stones; gems

**سَاتِن ، سَاتَگ** *sātin, sātag* فم | سَاتِل | نگَاه کَردَن ، نِگَهدَاشتَن protect, defend, take care of, guard

• **مَن آیرَا سَاتِین** | زَه هَغَه سَاتَم | مَن اُورَا نِگَاه میکُنَم I protect him

• **مَحمُود آیرَا سَاتِیت** | مَحمُود هَغَه سَاتِي | مَحمُود اُورَا نِگَاه میکُنَد Mahmud protects him

• **مَحمُود آیرَا سَاتِت** | مَحمُود هَغَه وُسَاتَه | مَحمُود اُورَا نِگَاه کَرد Mahmud protected him

**سَاتُوک** *sātōk* فَا | سَاتُونکَی | نِگَهبَان protector; defender; guard, watchman

Ende Wörterliste

Die Paschto-Version und die Dari-Version sind hier durch senkrechte Striche abgesetzt; sind beide Versionen identisch, so gibt es nur eine Angabe. Die Mühe einer zweispaltigen Darstellung haben wir uns nicht gemacht; sie bereitet aber keine besonderen Schwierigkeiten.

Das Schreiben von Makrodateien ist nicht einfach und erfordert viel Erfahrung, bis sich das gewünschte Ergebnis einstellt; allerdings kann diese Arbeit von einem Spezialisten unabhängig und parallel zur Erfassung der Eingabedaten erledigt werden, sobald einmal die abstrakte Syntax und ihre externe Darstellung festgeschrieben sind.

## 4.2 Linksläufiges Wörterbuch

Mit nahezu demselben Aufwand läßt sich auch eine linksläufige Darstellung derselben Wörterbuchdaten erreichen. Dazu verwenden wir nur eine andere Stildatei und lesen die Daten in einem linksläufigen Kontext ein.

```
% rzdict4d.tex
% Ausgabe als Liste linkslaeufig

\def \Kopf #1{% Beginn der Verarbeitung
  \LR{\large Linksl\"aufige W\"orterliste von Datei ``#1''}\par
  \bigskip }

\long\def \Process #1{% Rumpf verarbeiten, hier: ausfuehren
  #1 \relax }

\def \Fuss {% Ende der Verarbeitung
  \par \LR{Ende W\"orterliste}\par \endinput }

\def \Lm #1{\par\Large\setnashbf\setbalut #1 %
  \normalsize\setnash}                    % Lemma
\def \Va #1{${}^{#1}$ }                   % Varianten-Nr
\def \cf #1{$\leftarrow$ #1}              % Verweis
\def \Tr #1{\LR{\emph{#1}}}               % Transkription
\def \Gr #1{\setarab #1}                  % Grammatikangaben
\def \Be #1{$|$ \setbalut #1}             % Belutschi
\def \Pa #1{$|$ \setpashto #1}            % Pashto
\def \Da #1{$|$ \setfarsi #1}             % Dari
\def \En #1{\LR{#1}}                      % Englisch
\def \Ex #1{\par$\bullet$\large\setnashbf\setbalut #1%
  \normalsize\setnash}                    % Beispiel

\allowarab \Lm \allowarab \Va \allowarab \cf \allowarab \Tr
\allowarab \Gr \allowarab \Be \allowarab \Pa \allowarab \Da
\allowarab \En \allowarab \Ex
```

Jetzt wird der ganze Eintrag von ArabTEX gelesen und aufbereitet, und daher müssen wir in der Stildatei auch alle neu definierten Kommandos durch das Kommando `\allowarab` bei ArabTEX anmelden. Rechtsläufige Anteile werden jetzt Einschübe mittels `\LR`, und man sieht wie auch schon beim ersten Beispiel, daß die Übersetzungsangaben sprachabhängig interpretiert werden.

Linksläufige Wörterliste von Datei “Belutschi”

**سَا** [1] *sā* ن | أوربَنِي | جَو barley

**سَا** [2] ← سَاتِن

**سَاپ** *sāp* ص | صَاف clean, pure

**سَاپدِل** *sāpdil* ص | سپِینَی، سپِین زَڑی | صَاف دِل honest, sincere, frank

**سَات** *sāt* ن | گَانَه، زَہوَر | زیوَر، جَواهِرَات jewellery; precious stones; gems

**سَاتِن، سَاتگ** *sātin, sātag* فم | سَاتِل | نِگَهدَاشتَن، نِگَاه کَردَن protect, defend, take care of, guard

- **مَن آپِرَا سَاتِین** | زَه هَغَه سَاتَم | مَن اُورَا نِگَاه میکُنَم I protect him
- **مَحمُود آپِرَا سَاتِیت** | مَحمُود هَغَه سَاتِي | مَحمُود اُورَا نِگَاه میکُنَد Mahmud protects him
- **مَحمُود آپِرَا سَاتِت** | مَحمُود هَغَه وُسَاتَه | مَحمُود اُورَا نِگَاه کَرد Mahmud protected him

**سَاتُوک** *sātōk* فَا | سَاتُونکَی | نِگَهبَان protector; defender; guard, watchman

Ende Wörterliste

Überraschend ist hier der Zeilenumbruch: wenn der am Ende eines Eintrags, also ganz links, stehende englische Text nicht mehr auf die aktuelle Zeile paßt, so beginnt er am linken Rand, läuft nach rechts bis vor das letzte Wort in arabischer Schreibung, und setzt sich in der nächsten Zeile fort, die rechtsbündig abgelegt wird. Das sieht vielleicht befremdlich aus, ist aber nachweislich die einzig korrekte Lösung für das Einmischen rechtsläufiger Einschübe in einen linksläufigen Text[6].

[6] Donald E. Knuth and Pierre A. MacKay, *Mixing right-to-left texts with right-to-left texts*, TUGboat 8/1, 14–25 (1987)

## 4.3 Umwandlung in eine XML-Codierung

Entgegen unseren anfänglichen Erwartungen hat sich herausgestellt, daß sich unsere Wissensbasis aus dem angegebenen TEX-Format auch ohne großen Aufwand in eine Darstellung in XML (Extended Markup Language)[7] automatisch überführen läßt. Eine XML-Datei hat eine strenge Baumstruktur, die sich durch eine Grammatik in einer DTD (Document Type Declaration) beschreiben läßt; diese legt außer den in der abstrakten Syntax festgelegten inhaltlichen Elementen auch Regeln für deren konkrete Notation fest.

```
<!-- rzdict.dtd -->
<?xml version="1.0" encoding="UTF-8" ?>

<!ELEMENT dict (entry*)>
<!ATTLIST dict title CDATA #REQUIRED>

<!ELEMENT entry (variant?,(link|(trans,use?,vers*,example*)))>
<!ATTLIST entry lemma CDATA #REQUIRED>

<!ELEMENT variant EMPTY>
<!ATTLIST variant nr CDATA #REQUIRED>

<!ELEMENT link EMPTY>
<!ATTLIST link ref CDATA #REQUIRED>

<!ELEMENT trans (#PCDATA)>

<!ELEMENT use EMPTY>
<!ATTLIST use cat CDATA #REQUIRED>

<!ELEMENT vers (#PCDATA)>
<!ATTLIST vers lang (b|p|d|e) #REQUIRED>

<!ELEMENT example (vers*)>
<!ATTLIST example text CDATA #REQUIRED>
```

Zu jedem der Sprachelemente, die den Knoten in der Baumstruktur entsprechen, ist hier angegeben, welche weiteren Komponenten darin enthalten sein dürfen, welche Attribute vorhanden sind und wie diese heißen, und was jeweils der zulässige Wertebereich ist.

[7] http://www.w3.org/XML/

Um unsere Daten in XML umzuwandeln, verwenden wir wiederum eine Stildatei, die nunmehr etwas gründlicher als in den bisherigen Fällen die vorhandene Baumstruktur analysieren und berücksichtigen muß. Dies geschieht zum einen in den Makros `\Process` und `\Lm`, die zusammen einen gesamten Eintrag einlesen und als Baumknoten ausgeben, wobei das Lemma als Attribut abgelegt wird; zum anderen im Makro `\Ex` mit dem Hilfsmakro `\ExI`, das eine ganze Folge von Beispielen verarbeiten können muß. Das Makro `\ww` schreibt sein Textargument *verbatim* in eine Ausgabedatei, auch wenn darin TEX-Makros enthalten sind.

```
% rzdict2e.tex
% Ausgabe im XML-Format

\def \Kopf #1{% Beginn der Verarbeitung
  \ww {<?xml version="1.0"?>}
  \ww {<!DOCTYPE dict SYSTEM "rzdict.dtd">}
  \ww {<dict title="#1">}}

\long\def \Process #1{% Rumpf verarbeiten, hier: ausfuehren
  #1\relax\ww{</entry>}}

\def \Fuss {% Ende der Verarbeitung
  \ww {</dict>}}

\def \Lm #1{\ww{<entry lemma="#1">}}          % Eintrag
\def \Va #1{\ww{ <variant nr="#1"/>}}         % Variante
\def \Tr #1{\ww{ <trans>#1</trans>}}          % Transkription
\def \cf #1{\ww{ <link ref="#1"/>}}           % Verweis
\def \Gr #1{\ww{ <use cat="#1"/>}}            % Kategorie
\def \Be #1{\ww{  <vers lang="b">#1</vers>}}  % Belutschi
\def \Pa #1{\ww{  <vers lang="p">#1</vers>}}  % Paschto
\def \Da #1{\ww{  <vers lang="d">#1</vers>}}  % Dari
\def \En #1{\ww{  <vers lang="e">#1</vers>}}  % Englisch

\long\def \Ex #1#2\relax {%               % Start Beispiel
  \ww{<example text="#1">}\ExI #2\Ex\relax}

\long\def \ExI #1\Ex #2\relax {%          % Rumpf Beispiel
  #1\relax\ww{</example>}%                % abschliessen
  \cond{\ifx #2\empty}{}{\Ex #2\relax}\fi} % folgende Beispiele
```

Als Ergebnis der Umwandlung hat unsere Wissensbasis im XML-Format nun die folgende Gestalt:

```
<!-- rzdict1a.xml -->
<?xml version="1.0"?>
<!DOCTYPE dict SYSTEM "rzdict.dtd">
<dict title="Belutschi">
<entry lemma="sA">
 <variant nr="1"/>
 <trans>s\=a</trans>
 <use cat="n"/>
 <vers lang="p">Orbe,sI</vers>
 <vers lang="d">^gaw</vers>
 <vers lang="e">barley</vers>
</entry>
<entry lemma="sA">
 <variant nr="2"/>
 <link ref="sAtin"/>
</entry>
<entry lemma="sAp">
 <trans>s\=ap</trans>
 <use cat=".s"/>
 <vers lang="d">.sAf</vers>
 <vers lang="e">clean, pure</vers>
</entry>
<entry lemma="sApdil">
 <trans>s\=apdil</trans>
 <use cat=".s"/>
 <vers lang="p">spInY, spIn ze,rY</vers>
 <vers lang="d">.sAf dil</vers>
 <vers lang="e">honest, sincere, frank</vers>
</entry>
<entry lemma="sAt">
 <trans>s\=at</trans>
 <use cat="n"/>
 <vers lang="p">gA.nah, zEwar</vers>
 <vers lang="d">zEwar, ^gawAhirAt</vers>
 <vers lang="e">jewellery; precious stones; gems</vers>
</entry>
<entry lemma="sAtin, sAtag">
```

```
 <trans>s\=atin, s\=atag</trans>
 <use cat="fm"/>
 <vers lang="p">sAtil</vers>
 <vers lang="d">nigahdA^stan, nigAh kardan</vers>
 <vers lang="e">protect, defend, take care of, guard</vers>
<example text="man AyirA sAtIn">
 <vers lang="p">zeh ha.geh sAtem</vers>
 <vers lang="d">man UrA nigAh mEkunam</vers>
 <vers lang="e">I protect him</vers>
</example>
<example text="ma.hmUd AyirA sAtIt">
 <vers lang="p">ma.hmUd ha.geh sAtI</vers>
 <vers lang="d">ma.hmUd UrA nigAh mEkunad</vers>
 <vers lang="e">Mahmud protects him</vers>
</example>
<example text="ma.hmUd AyirA sAtit">
 <vers lang="p">ma.hmUd ha.geh wusAteh</vers>
 <vers lang="d">ma.hmUd UrA nigAh kard</vers>
 <vers lang="e">Mahmud protected him</vers>
</example>
</entry>
<entry lemma="sAtOk">
 <trans>s\=at\=ok</trans>
 <use cat="fA"/>
 <vers lang="p">sAtUnkY</vers>
 <vers lang="d">nigahbAn</vers>
 <vers lang="e">protector; defender; guard, watchman</vers>
</entry>
</dict>
```

Diese Darstellung kann nun mit XML-Werkzeugen weiter verarbeitet werden, soweit solche verfügbar sind. Allerdings ist uns bislang kein System bekannt, das in der Lage wäre, ausgehend von einer XML-Darstellung die erweiterte arabische Schrift darzustellen, sodaß man für eine graphische Aufbereitung doch wieder irgendwann auf TEX und ArabTEX wird zurückgreifen müssen. Wir zeigen im Folgenden einige Möglichkeiten auf, dies ohne allzugroßen Aufwand zu erreichen.

# 5 Verarbeitung der XML-Darstellung

Ausgehend von der oben gewonnenen XML-Darstellung kann man nun versuchen, für Zwecke der graphischen Aufbereitung auf die TEX-Darstellung der Wissensbasis zurückzugehen, für die ja bereits Aufbereitungsroutinen vorhanden sind. Es liegt nahe, die zuletzt verwendete Stildatei, welche die Umwandlung vom TEX-Format ins XML-Format leistet, gewissermaßen „umzudrehen" und eine Transformationsroutine zu schreiben, welche die XML-Auszeichnungen analysiert, und daraus wieder die TEX-Darstellung erzeugt und diese in eine Datei ausschreibt.

## 5.1 Umwandlung mittels einer XSL-Transformation

Die oben gewonnene XML-Darstellung läßt sich nun mit passenden XML-Prozessoren weiterverarbeiten. Als Beispiel geben wir hier eine Transformation an, die in der Sprache XSL (Extended Stylesheet Language)[8] geschrieben ist und die wir mit dem Werkzeug XALAN[9] erfolgreich ausführen konnten.

```
<!-- rzdict1a.xsl -->
<?xml version="1.0"?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
<xsl:output method="text" indent="no" encoding="UTF-8"/>
 <xsl:template match="/">
   <xsl:apply-templates/>
 </xsl:template>
 <xsl:template match="dict">
   <xsl:value-of select="@title"/>
   <xsl:apply-templates/>
 </xsl:template>
 <xsl:template match="entry">
   \Lm {<xsl:value-of select="@lemma"/>}
   <xsl:apply-templates/>
 </xsl:template>
 <xsl:template match="variant">
   \Va {<xsl:value-of select="@nr"/>}
 </xsl:template>
```

[8] http://www.w3.org/Style/XSL/

[9] http://xml.apache.org/

```
 <xsl:template match="link">
   \cf {<xsl:value-of select="@ref"/>}
 </xsl:template>
 <xsl:template match="trans">
   \Tr {<xsl:apply-templates/>}
 </xsl:template>
 <xsl:template match="use">
   \Gr {<xsl:value-of select="@cat"/>}
 </xsl:template>
 <xsl:template match="vers[@lang='b']">
   \Be {<xsl:apply-templates/>}
 </xsl:template>
 <xsl:template match="vers[@lang='p']">
   \Pa {<xsl:apply-templates/>}
 </xsl:template>
 <xsl:template match="vers[@lang='d']">
   \Da {<xsl:apply-templates/>}
 </xsl:template>
 <xsl:template match="vers[@lang='e']">
   \En {<xsl:apply-templates/>}
 </xsl:template>
 <xsl:template match="example">
   \Ex {<xsl:value-of select="@text"/>}
   <xsl:apply-templates/>
 </xsl:template>
</xsl:stylesheet>
```

Das Ergebnis sind wiederum unsere Ausgangsdaten, nunmehr in der TEX-Darstellung. Wir verlieren keine Information dabei.

## 5.2 Umwandlung nach TEX mittels TEX

XML-Werkzeuge stehen nicht überall zur Verfügung. Daher ist es beruhigend zu wissen, daß die Umwandlung in die TEX-Darstellung auch mittels TEX selbst geleistet werden kann unter Verwendung eines (wenig bekannten) Kunstgriffes: man kann die einleitende spitze Klammer `<` der XML-Auszeichnungen als Steuerzeichen umdefinieren, sodaß die Auszeichnungen wiederum als TEX-Kommandos gedeutet werden, und man definiert ihre Auswirkungen entsprechend. Allerdings hat sich nun die globale Struktur unserer Liste formal geändert, daher müssen wir unser Treiberprogramm anpassen:

```
% rzdict3c.tex
% Treiber fuer Woerterbuch in XML-Darstellung
% als <dict titel="Titel"> Eintrag* </dict>
% jeder Eintrag hat das Format <entry ... </entry>
\catcode `\< 0                           % < als Kommandoanfang

\long\def <dict title="#1">{%          Eintraege verarbeiten
  \Kopf {#1}\Eintrag }

\long\def \Eintrag #1<entry #2</entry>#3{%  Eintrag einlesen
  <entry #2</entry>%          ausfuehren, evtl. weitermachen
  \cond{\ifx #3\/}{\EintragI }{\Eintrag #3}\fi }

\def \EintragI dict>{\Fuss }             % </dict> wegnehmen
```

In diesem Treiberprogramm tritt jetzt die früher verwendete Routine `\Process` nicht mehr auf. Sie wird überflüssig, weil nun das Kommando `<entry` mit dem gesamten Inhalt eines Eintrags versorgt wird; so kann eine individuelle Aufbereitung, falls gewünscht, ganz innerhalb der jeweiligen Stildatei realisiert werden.

Mittels einer solchen Stildatei können wir nun die Dateninhalte im TEX-Format in eine Ausgabedatei schreiben:

```
% rzdict2f.tex
% Umwandlung XML-Format nach \TeX-Ausgabe
\catcode `\< 0                           % < als Kommandoanfang

\def \Kopf #1{%                          % Titel
  \ww{\Dict{#1}}}

\def \Fuss {%                            % Ende Verarbeitung
  \ww {\Lm{*}}}

\long\def <?#1?>{}                       % XML Kopf weg
\long\def <!#1>{}                        % Kommentare weg

\def <entry lemma="#1">#2</entry>{%      % Eintrag
  \ww{\Lm{#1}}#2\relax \ww{}}
\def <variant nr="#1"/>{\ww{ \Va{#1}}}   % Varianten-Nr
\def <link ref="#1"/>{\ww{ \cf{#1}}}     % Verweis
\def <trans>#1</trans>{\ww{ \Tr{#1}}}    % Transkription
```

```
\def <use cat="#1"/>{\ww{ \Gr{#1}}}      % Grammatikangaben
\def <vers lang="#1">#2</vers>{%         % Uebersetzung
  \ifx #1b\ww{ \Be{#2}}\else             % Belutschi
  \ifx #1p\ww{ \Pa{#2}}\else             % Paschto
  \ifx #1d\ww{ \Da{#2}}\else             % Dari
  \ifx #1e\ww{ \En{#2}}\else             % Englisch
  \fi\fi\fi\fi}
\def <example text="#1">#2</example>{%   % Beispiel
  \ww{\Ex{#1}}#2}
```

Das Ergebnis der Umwandlung, das wir hier zur Kontrolle einlesen, sieht nun nahezu wie unsere Ausgangsdaten aus:

```
% rzdict2a.dat
\Dict {Belutschi}
\Lm {sA}
 \Va {1}
 \Tr {s\=a}
 \Gr {n}
 \Pa {Orbe,sI}
 \Da {^gaw}
 \En {barley}

\Lm {sA}
 \Va {2}
 \cf {sAtin}

\Lm {sAp}
 \Tr {s\=ap}
 \Gr {.s}
 \Da {.sAf}
 \En {clean, pure}

\Lm {sApdil}
 \Tr {s\=apdil}
 \Gr {.s}
 \Pa {spInY, spIn ze,rY}
 \Da {.sAf dil}
 \En {honest, sincere, frank}

\Lm {sAt}
```

```
 \Tr {s\=at}
 \Gr {n}
 \Pa {gA.nah, zEwar}
 \Da {zEwar, ^gawAhirAt}
 \En {jewellery; precious stones; gems}

\Lm {sAtin, sAtag}
 \Tr {s\=atin, s\=atag}
 \Gr {fm}
 \Pa {sAtil}
 \Da {nigahdA^stan, nigAh kardan}
 \En {protect, defend, take care of, guard}
\Ex {man AyirA sAtIn}
 \Pa {zeh ha.geh sAtem}
 \Da {man UrA nigAh mEkunam}
 \En {I protect him}
\Ex {ma.hmUd AyirA sAtIt}
 \Pa {ma.hmUd ha.geh sAtI}
 \Da {ma.hmUd UrA nigAh mEkunad}
 \En {Mahmud protects him}
\Ex {ma.hmUd AyirA sAtit}
 \Pa {ma.hmUd ha.geh wusAteh}
 \Da {ma.hmUd UrA nigAh kard}
 \En {Mahmud protected him}

\Lm {sAtOk}
 \Tr {s\=at\=ok}
 \Gr {fA}
 \Pa {sAtUnkY}
 \Da {nigahbAn}
 \En {protector; defender; guard, watchman}

\Lm {*}
```

Unser Mechanismus reproduziert, ausgehend von der automatisch erzeugten XML-Datei, unsere ursprünglichen TEX-Eingabedaten Zeichen für Zeichen bis auf Kommentare und evtl. zusätzliche Zwischenräume und Zeilenwechsel, auf die es aber nicht ankommt. Damit kann die endgültige Festlegung auf eine der beiden vorgeschlagenen Darstellungen für unsere Wissensbasis auf einen späteren Zeitpunkt verschoben und auch jederzeit noch korrigiert werden.

## 5.3 Kombinierte Umwandlung und Aufbereitung

Es gibt aber noch eine bei weitem elegantere Möglichkeit. Zufällig und glücklicherweise gibt es zwischen den Namen der XML-Auszeichnungen und unseren TEX-Attributmarkierungen fast keine Überlappung, und wir können uns daher darauf beschränken, die XML-Markierungen einfach in die entsprechenden TEX-Kommandos samt der passenden Versorgung umzusetzen. Das früher verwendete Hilfsmakro `\ww` kann nun so umdefiniert werden (siehe Anhang), daß es sein Argument nicht mehr in eine Datei ausgibt, sondern in einer geeigneten Umgebung direkt ausführt.

Aktivieren wir nun noch zusätzlich eine unserer früheren Umwandlungen, etwa diejenige zur rechtsläufigen graphischen Ausgabe, so bekommen wir den kombinierten Effekt, also hier die rechtsläufige Ausgabe nun ausgehend von der XML-Darstellung, praktisch gratis.

Rechtsläufige Wörterliste von Datei “Belutschi”

**سَا** 1 *sā* ن | أورتَبِني | جَو barley

**سَا** 2 → سَاتِن

**سَاپ** *sāp* ص | صَاف clean, pure

**سَاپدِل** *sāpdil* ص | سپِینَی، سپِین زَرَہی | صَاف دِل honest, sincere, frank

**سَات** *sāt* ن | گَانَه، زَہوَر | زیوَر، جَوَاهِرَات jewellery; precious stones; gems

**سَاتِن، سَاتگ** *sātin, sātag* فِم | سَاتِل | نِگهدَاشتَن، نِگَاه کَردَن protect, defend, take care of, guard

• **مَن آیرَا سَاتِین** | زَه هَغَه سَاتَم | مَن اُورَا نِگَاه میکُنَم I protect him

• **مَحمُود آیرَا سَاتِیت** | مَحمُود هَغَه سَاتِي | مَحمُود اُورَا نِگَاه میکُنَد Mahmud protects him

• **مَحمُود آیرَا سَاتِت** | مَحمُود هَغَه وُسَاتَه | مَحمُود اُورَا نِگَاه کَرد Mahmud protected him

**سَاتُوک** *sātōk* فَا | سَاتُونکَی | نِگهبَان protector; defender; guard, watchman

Ende Wörterliste

Wir haben damit geradezu einen „Baukasten“ von Umwandlungsroutinen erhalten, die wir soweit sinnvoll weitgehend kombinieren können. Wir benötigen für jede Kombination zum Aufruf ein Steuerdokument, das man *ad hoc* schreiben kann, und das nicht viel mehr tun muß, als eine passende Auswertungsumgebung einzustellen, und die benötigten Makrodateien sowie die Eingangsdaten einzulesen und dabei zu aktivieren. Als sinnvolle Abrundung wäre nun noch für jedes verwendete Ablageformat ein Paket denkbar, das nichts weiter tut, als die Daten im gleichen Format wieder in eine Datei zu schreiben. So kann man jederzeit Zwischenstufen der Umwandlungen sicherstellen.

# 6 Sortieren

Unser Komponentenkonzept hat sich als überraschend tragfähig herausgestellt, und es läßt sich noch stark erweitern. So will man womöglich später weitere Versionen des Wörterbuches erstellen, die von jeweils einer anderen Sprache ausgehen und auch nach den entsprechenden Feldern sortiert sind; aber auch schon für die weitere Pflege der Daten ist eine Möglichkeit zum Sortieren der Datensätze fast unverzichtbar.

Die üblicherweise verfügbaren Sortierprogramme können das nicht direkt leisten, weil die Wörter und Wortformen hier in einer Umschrift angegeben sind, die mit der gewünschten Sortierfolge nichts zu tun hat; außerdem setzen sie gewöhnlich Datensätze in festem Format voraus, und davon kann hier nicht die Rede sein. Wir besitzen jedoch bereits ein (selbst entwickeltes) universelles Sortierprogramm[10], das in der Lage ist, Datensätze in einem Format zu verarbeiten, das zu dem hier vorgeschlagenen nahe verwandt ist.

Die Grundidee ist, einen alphanumerischen Sortierschlüssel als ein zusätzliches Attribut bei den Einträgen der Wissensbasis jeweils mit aufzunehmen, der bei der Auswertung ignoriert werden kann, und diesen Sortierschlüssel jeweils aus dem für die Sortierung maßgeblichen Teilfeld zu berechnen. Wir nutzen dazu (bislang unpublizierte und undokumentierte) Eingriffsmöglichkeiten im System ArabTEX, über die man auf die interne Darstellung der arabischen Schriftform zugreifen kann, und aus dieser läßt sich ein brauchbarer Sortierschlüssel ableiten. Wir generieren den Schlüssel in unserem Beispiel aus dem Lemma und machen ihn bei Homographen über die Variante eindeutig. Um ihn zu den Datensätzen zuzufügen und somit eine zum Sortieren geeignete Datei zu erstellen, genügt eine weitere Stildatei:

[10] `http://pcbs13.informatik.uni-stuttgart.de/ifi/bs/research/mldict/rzsort.tex`

```
% rzdict2h.tex
% Ausgabe mit Sortierschl"ussel aus dem \Lm-Feld
% notfalls eindeutig ueber Variante

\def \Kopf #1{% Beginn der Verarbeitung
  \ww {\Dict{#1}}}

\long\def \Process #1{%                        Rumpf verarbeiten
  \Select \Lm #1\relax              % Feld \Lm herausgreifen
  \Select \Va #1\relax              % Feld \Va herausgreifen
  \xpa\wkey \xpa{\thekey}%             % Schluessel ausgeben
  \Rumpf #1\relax }                         % Rumpf ausgeben

\def \Fuss {% Ende der Verarbeitung
  \ww {\key{*}}}

\long\def \Select #1#2\relax {% Feld #1 aus Rumpf #2 ausfuehren
  \begingroup \long\def \lookup ##1#1##2##3\relax
    {\endgroup #1{##2}}%
  \lookup #2#1{}\relax }

\def \Lm #1{%                        % Schluessel aus Lemma
  \makeKey {#1}}                     % extern, setzt \thekey

\def \Va #1{\def \temp {#1}%         % Schluessel eindeutig
  \cond{\ifx \temp\empty}{}%         % ueber Variante
  {\edef \thekey {\thekey .#1}}\fi}  % falls noetig

\def \wkey #1{%                      % Schluessel absetzen
  \ww {\key {#1}}}

\long\def \Rumpf #1 #2\relax{%       % Rumpf zerlegen
  \ww { #1}%                         % erstes Feld
  \cond{\ifx #2\par }{\ww {}}%
    {\Rumpf #2\relax }\fi}% Rest
```

Mit dem Makro `\Select` kann man den Wert eines beliebigen Attributes aus dem Rumpf auslesen und das zugeordnete Kommando ausführen; es kann auch für andere Anwendungen nützlich sein. Die Hauptarbeit in dieser Stildatei leistet das Makro `\makeKey`, das die eigentliche Schlüsselberechnung durchführt und im Anhang wiedergegeben ist.

Angewandt auf unsere Originaldaten ergibt sich die neue Datei:

```
% rzdict3.dat
\Dict {Belutschi}
\key {2611.1}
 \Lm {sA}
 \Va {1}
 \Tr {s\=a}
 \Gr {n}
 \Pa {Orbe,sI}
 \Da {^gaw}
 \En {barley}

\key {2611.2}
 \Lm {sA}
 \Va {2}
 \cf {sAtin}

\key {261113}
 \Lm {sAp}
 \Tr {s\=ap}
 \Gr {.s}
 \Da {.sAf}
 \En {clean, pure}

\key {2611132140}
 \Lm {sApdil}
 \Tr {s\=apdil}
 \Gr {.s}
 \Pa {spInY, spIn ze,rY}
 \Da {.sAf dil}
 \En {honest, sincere, frank}

\key {261114}
 \Lm {sAt}
 \Tr {s\=at}
 \Gr {n}
 \Pa {gA.nah, zEwar}
 \Da {zEwar, ^gawAhirAt}
 \En {jewellery; precious stones; gems}
```

```
\key {26111442}
 \Lm {sAtin, sAtag}
 \Tr {s\=atin, s\=atag}
 \Gr {fm}
 \Pa {sAtil}
 \Da {nigahdA^stan, nigAh kardan}
 \En {protect, defend, take care of, guard}
 \Ex {man AyirA sAtIn}
 \Pa {zeh ha.geh sAtem}
 \Da {man UrA nigAh mEkunam}
 \En {I protect him}
 \Ex {ma.hmUd AyirA sAtIt}
 \Pa {ma.hmUd ha.geh sAtI}
 \Da {ma.hmUd UrA nigAh mEkunad}
 \En {Mahmud protects him}
 \Ex {ma.hmUd AyirA sAtit}
 \Pa {ma.hmUd ha.geh wusAteh}
 \Da {ma.hmUd UrA nigAh kard}
 \En {Mahmud protected him}

\key {2611144537}
 \Lm {sAtOk}
 \Tr {s\=at\=ok}
 \Gr {fA}
 \Pa {sAtUnkY}
 \Da {nigahbAn}
 \En {protector; defender; guard, watchman}

\key {*}
```

Aus der so erzeugten Eingabedatei generiert das Sortierprogramm eine nach den Schlüsseln sortierte Ausgabedatei im selben Format. Sie kann nun auf verschiedenartige Weise weiterverarbeitet werden: man kann die Schlüssel abtrennen und so auf das bisherige Format zurückgehen, oder man kann das Treiberprogramm, das man für die Abtrennung braucht, auch für weitere Verarbeitungen verwenden und die Schlüssel einfach stehen lassen. Das Treiberprogramm isoliert nun die Einträge an Hand der Schlüssel; es ist nur eine Abwandlung unserer ersten Version:

```
% rzdict3d.tex
% Treiber fuer Woerterbuch als \Dict{Titel} Eintrag* Datenende
% jeder Eintrag beginnt mit \key
% Datenende: \key{*}
% \Process verarbeitet den Eintrag

\long\def \Dict #1#2\key {% Eintraege verarbeiten
\Kopf {#1}\Eintrag \key }

\long\def \Eintrag \key #1#2\key #3{%  ganzen Eintrag einlesen
\Process {\key{#1}#2}%    Rumpf ausfuehren, evtl. weitermachen
\cond{\ifx *#3}{\Fuss }{\Eintrag \key {#3}}\fi }
```

Wenn man die Schlüssel bestehen läßt, so kann man bei der weiteren Pflege der Daten Korrekturen und Aktualisierungen mit geringem Aufwand einmischen. Wir führen dies hier nicht weiter aus.

# 7 Ausblick

Wir haben einen Satz von Bausteinen zur Unterstützung der Verwaltung und Umwandlung einer Wissensbasis am Beispiel eines mehrsprachigen Wörterbuches vorgestellt. Unser Bericht ist recht umfangreich geraten, und dies hat einen guten Grund: die vorgestellten Bausteine werden hier nicht nur aus den Originalquellen eingelesen und somit zuverlässig dokumentiert, sondern sie werden bei der Formatierung des Berichtes auch tatsächlich durchlaufen (mit Ausnahme der XSLT-Transformation und des externen Sortierprogramms). So überzeugen wir uns gleichzeitig von der korrekten Arbeitsweise. Die Geschwindigkeit beeindruckt uns: auf einem mittelschnellen PC brauchen wir nur 7 Sekunden.

Die hier vorliegende Implementierung[11] ist als ein Prototyp zu verstehen; wir haben ihre Brauchbarkeit bislang nur an einem Spielbeispiel erprobt. Bei der Anwendung auf umfangreichere Originaldaten mögen sich noch Änderungen und Erweiterungen ergeben, ebenso bei Änderung der grundsätzlichen Struktur der Eingangsdaten; in Folge des stark modularen Aufbaues unserer Bausteine dürften notwendige Anpassungen aber lokal und mit geringem Aufwand durchführbar sein.

[11] `http://pcbs13.informatik.uni-stuttgart.de/ifi/bs/research/mldict.htm/`

# Anhang

## A Berechnung von Sortierschlüsseln

Das folgende Programmstück setzt voraus, daß ArabTEX geladen ist; es verwendet einige undokumentierte Schnittstellen, und es steht hier nur zum Zweck der Dokumentation.

```
% rzdict2i.tex
% berechne Sortierschluessel
% Aufruf: \makeKey {arabischer Text}
% Ergebnis: \thekey

\def \makeKey #1{% generate key
\genkey #1 \relax }

\def \genkey #1 #2\relax {% take only the first word
{\a@@writefalse \tracingarab 0 % disable Arabic output
\a@rabwrite{#1}% process the Arabic lemma by ArabTeX
\gdef \thekey {}\xpa \addtokey \the\sv@revwd .*}}

\def \addtokey #1#2*{% from the carrier, forget the modifiers
\ifx .#1\else \ReturnFi % append to key
{\ifcat a#1%
\xdef \thekey {\thekey \csname as@\string #1\endcsname }%
\fi \addtokey }\fi }

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Berechne eine Sortierfolge

\newcount \nkey

\def \defsort #1#2{% define #1 by sorting sequence #2
\def \thesort {}\nkey = 10
\gensort #2 \end \relax \let #1=\thesort }

\def \gensort #1 #2\relax {% separate words
\makesort {#1}%
\ifx \end #2\else \ReturnFi % append to key
{\gensort #2\relax }\fi }
```

```
\def \makesort #1{% extend \thesort
{\a@@writefalse \tracingarab 0
\a@rabwrite {#1}% process the word by ArabTeX
\xpa \addtosort \the\sv@revwd .*}}

\def \addtosort #1#2*{% from the carrier, forget the modifiers
\ifx .#1\else \ReturnFi % append to key
{\ifcat a#1\global\advance \nkey 1
\edef \thecmd {\def\xpa\nxp\csname as@#1\endcsname{\the\nkey}}%
\xpa\a@extend \xpa\thesort \xpa{\thecmd }%
\fi \addtosort }\fi }

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Vorgabe einer Sortierfolge

\defsort \arabsort {% arabic sorting sequence
'a bpt_t,t ^g.h_hc,c^c d_d,d rz.r,r,z^z s^s,s
.s.d .t.z `.g fqv kg^n l^l m n.n hT w yY}

\arabsort % Sortierfolge einstellen

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Das Makro `\makeKey` berechnet aus dem ersten arabischen Wort seiner Eingabe einen alphanumerischen Sortierschlüssel und hinterlegt ihn im globalen Makro `\thekey`.

Über das Makro `\defsort` kann der Benutzer eine Sortierfolge vorgeben und ihr einen Kommandonamen zuordnen. Derzeit ist die Standard-Sortierfolge für Arabisch eingestellt; Erweiterung und Umstellung auf andere Sprachen ist einfach, sobald die Sortierfolge bekannt ist.

# B Hilfsmakros und Umgebungen

Bei der Aufbereitung dieses Berichtes haben wir alle angesprochenen Routinen tatsächlich durchlaufen und sie somit gleichzeitig getestet. Dabei mußten wir aber gelegentlich spezielle Umgebungen einstellen und haben außerdem einige Hilfsmakros verwendet. Wir geben sie hier an; damit ist das gesamte Material an einer Stelle zu finden.

## B.1 Hilfsmakros

```
% rzmac.sty
% Hilfsmakros fuer rzdict.tex
% teilweise Erweiterungen fuer ArabTeX
% 27.08.2002
% Klaus Lagally
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\a@ident {rzmac.sty}
{1.0 auxiliary commands for rzdict.tex } {27.08.2002}
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

\newwrite \outfile

\long\def \ww #1{\long\def \temp {#1}%
\xpa\wwI \xpa{\xpa\split\meaning\temp}}

\long\def \wwI #1{\immediate\write \outfile {#1}}

\def \split #1>{}

\let \ital \/

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\endinput
```

Wir öffnen einen Ausgabestrom und stellen die originale Bedeutung des TEX-Standardmakros `\/` sicher, das wir zwischenzeitlich für die XML-Auswertung umdefinieren müssen.

Das Makro `\ww` schreibt sein Argument unverändert, ohne Makroexpansionen, in die Ausgabedatei.

## B.2 Umgebung zur rechtsläufigen Ausgabe (Abschnitt 4.1)

```
\begin{quotation}
\parskip 2mm
\parindent -5mm
\input rzdict3b.tex
\input rzdict4c.tex
\input rzdict1.dat
\end{quotation}
```

Wir stellen einige Layout-Parameter um und lesen Treiberdatei, Stildatei und Daten ein. Sonst gibt es hier keine Besonderheiten.

## B.3 Linksläufige Ausgabe von TEX aus (Abschnitt 4.2)

```
\begin{quotation}
\parindent -5mm
\input rzdict3b.tex
\input rzdict4d.tex
\begin{RLtext}
\input rzdict1.dat
\end{RLtext}
\end{quotation}
```

Hier dürfen wir nur die Daten selbst innerhalb der `RLtext`-Umgebung einlesen, sonst ergeben sich Kollisionen mit ArabTEX.

## B.4 Umwandlung von TEX nach XML (Abschnitt 4.3)

```
\immediate\openout \outfile=rzdict1a.xml \relax
\ww {<!-- rzdict1a.xml -->}
\input rzdict3b.tex
\input rzdict2e.tex
\input rzdict1.dat
\immediate\closeout \outfile \relax
```

Wir schreiben den Dateinamen als Kommentar in die Ausgabedatei.

## B.5 Umwandlung von XML nach TEX (Abschnitt 5.2)

```
\immediate\openout \outfile=rzdict2a.dat \relax
{\catcode `\% 12 \ww {% rzdict2a.dat}}
\input rzdict3c.tex
\input rzdict2f.tex
\input rzdict1a.xml
\immediate\closeout \outfile \relax
```

Um das Prozentzeichen ausgeben zu können, müssen wir zwischenzeitlich seine Sonderbedeutung als Kommentarzeichen abschalten.

## B.6 Rechtsläufige Ausgabe von XML aus (Abschnitt 5.3)

```
\begin{quotation}
\parskip 2mm
\parindent -5mm
\def \ww #1{\catcode `\< 12 \let \/=\ital #1 \catcode `\< 0 }
\input rzdict3c.tex
\input rzdict2f.tex
\input rzdict4c.tex
\input rzdict1a.xml
\end{quotation}
```

Das Kommando `\ww` führt die Attribut-Kommandos in einer Umgebung aus, in der die Symbole `<` und `\/` zwischenzeitlich ihre ursprüngliche Bedeutung aus TEX/LATEX haben.

## B.7 Umgebung zur Schlüsselerzeugung (Abschnitt 6)

```
\immediate\openout \outfile=rzdict3.dat \relax
{\catcode `\% 12 \ww {% rzdict3.dat}}%
\input rzdict3b.tex
\input rzdict2i.tex
\input rzdict2h.tex
\input rzdict1.dat
\immediate\closeout \outfile \relax
```

Keine Besonderheiten.

# Inhaltsverzeichnis