7 Variablen und Bindungen

7.1 Motivation

Die Erstellung eines größeren Dokuments erfolgt im allgemeinen über einen längeren Zeitraum. Am Anfang sind Textteile und viele Festlegungen bezüglich der Darstellung noch offen. Eventuell stehen eine häufig vorkommende Bezeichnung oder sogar ein ganzer Textblock noch nicht fest. Vielleicht kommt solch ein Textblock in vielen verschiedenen „Varianten“ vor, die aus dem gleichen Ursprungstext durch Einsetzen von Textstücken hervorgegangen sind. Es wäre mühselig, wenn in solchen Fällen alle Vorkommen solcher Bezeichnungen bzw. Textblöcke im Dokument einzeln durch den Autor erstellt bzw. im Falle einer Änderung gefunden und geändert werden müßten. Deshalb ist es hilfreich, einen Namen für solche Textblöcke einführen zu können. Ein Vorkommen eines solchen Namens wird dann durch den von ihm bezeichneten Textblock ersetzt. Alle Veränderungen des Textblocks im Laufe der Entwicklung des Dokuments können dann an einer zentralen Stelle durchgeführt werden. Das ist die Stelle, an der der Name für den Textblock eingeführt wurde.

Andererseits gibt es Dokumentsorten mit vielen festen Bestandteilen. Ein Brief hat z. B. eine Absenderadresse, eine Empfängeradresse, eine Betreffzeile, eine Anrede, den eigentlichen Brieftext und am Ende eine Grußformel. Es gibt eine Reihe von Möglichkeiten, diese Bestandteile auf dem Briefpapier anzuordnen. Man kann sich also ein Briefformat so vorstellen, daß diese Bestandteile, mit einem Namen versehen, absolut oder relativ zueinander auf dem Briefpapier plaziert sind. Belegt man diese Namen mit Werten, also etwa den Namen *Absender* mit

> Fachbereich Informatik
> Universität des Saarlandes
> 66041 Saarbrücken,

so wird vermutlich rechts oben ein rechteckiger Kasten von passender Größe mit dem entsprechenden Inhalt angeordnet.

Der Satzspiegel, der Zeichensatz, der Zeilenabstand sowie die laufenden Kopf- und Fußzeilen stehen beim Erstellen des Dokuments eventuell noch nicht fest. Ob Formeln linksbündig oder zentriert gesetzt werden, ist noch unklar. Bei einem Buchprojekt legt der Verlag erst spät fest, ob Kapitel auf einer neuen, rechten Seite beginnen und ob es einen Index gibt und wohin er kommt. Die Autoren sollten ihr Dokument mit vorläufigen Festlegungen erstellen können und die endgültigen Einstellungen erst dann vornehmen, wenn sie ihnen bekannt sind.

Jedes halbwegs mächtige Dokumentverarbeitungssystem bietet einen Mechanismus an, mit dem diese Probleme zumindest teilweise gelöst werden können. Es gibt *Variablen*, das sind die oben erwähnten Namen, die durch *Definitionen* an Werte gebunden werden können. Diese Werte können, wie in den obigen Beispielen gezeigt, Texte sein, die dann für Vorkommen der Variablen in das Dokument eingesetzt werden, oder auch Zahlenwerte, die das endgültige Aussehen des Dokuments beeinflussen. Wichtig ist hier der Unterschied zwischen einem Wert und der Variablen, mit der man ihn benennt.

Mehrere Konstrukte haben mit Variablen zu tun. Die *Deklaration* einer Variablen macht diese dem dokumentverarbeitenden System bekannt. Eine *Definition* bindet sie an einen Wert. Dabei wird eventuell eine vorherige Bindung überschrieben. Oft treten Deklaration und initiale Definition zusammen auf. Die Vorkommen von Variablen in Deklarationen und Definitionen werden *definierende Vorkommen* genannt. Daneben gibt es noch *angewandte Vorkommen*, wo auf den an die Variable gebundenen Wert zugegriffen wird. Die Menge aller Variablenbindungen wird hier in Anlehnung an die übliche Programmiersprachenterminologie als *Umgebung* bezeichnet.

7.2 Arten von Variablen

Die obigen Beispiele haben gezeigt, daß Variablen mehrere Funktionen haben können:

- Sie können für Umbruchparameter stehen, etwa den Zeichensatz, den Zeilenabstand, Randgrößen usw. Diese Variablen sind im System fest „eingebaut“. Man nennt sie deshalb *systemdefinierte* Variablen. Meist haben sie vordefinierte Werte, die vom Benutzer (eventuell sogar mehrfach) geändert werden können. Die „aktuellen“ Werte der Variablen beeinflussen jeweils Umbruchentscheidungen.
- Variablen können auch für feste Dokumentbestandteile stehen, etwa für die Inhalte von Kopf- und Fußzeilen, für feste Bestandteile von Tabellen und Abbildungen, für laufende Überschriften und Seitennummern und für das aktuelle Datum. Auch solche Variablen sind meist systemdefiniert. Teilweise können sie vom Benutzer definiert bzw. umdefiniert werden, wie etwa „Abbildung“, „Abb.“, oder „Fig.“ in Bildunterschriften, teilweise werden sie vom System automatisch gesetzt, etwa das Datum oder Seitenzahlen und laufende Überschriften. Letztere können z. B. aus den Kapitelüberschriften bestehen.
- Variablen können als Stellvertreter in Texten stehen. Dies erlaubt das einfache Ändern von häufig auftretenden Texten. Wann immer das System beim Umbrechen oder Darstellen des Dokuments auf eine solche Variable trifft, so wird ihr aktueller Wert für sie eingesetzt. Solche Variablen kann der Benutzer selbst definieren und dann an Werte binden.

7.2.1 Variablen in LaTeX

In einem Batch-System wie LaTeX sind Deklarationen und Definitionen von Variablen Teile der Dokumentbeschreibung. Da sie selbst keinen Ausgabetext erzeugen, können sie im Prinzip überall stehen; meist werden sie am Anfang der Dokumentbeschreibung oder in einer eigenen Datei gesammelt.

In LaTeX gibt es mehrere Klassen von Variablen, die verschieden deklariert, definiert und benutzt werden können. Die wichtigsten Klassen, die im folgenden besprochen werden, sind *Zählervariablen*, *Längenvariablen* und *Textvariablen* (*Makros*); es gibt aber noch mehr.

Zählervariablen

Zählervariablen sind Variablen, die an ganze Zahlen gebunden werden können. Wichtige vordefinierte Zählervariablen sind `page` für die Seitennummer sowie `chapter`, `section` usw. für die laufende Nummer von Kapiteln, Abschnitten usw. Wie an diesen Beispielen zu erkennen ist, sind die Namen von Zählervariablen einfache Buchstabenfolgen ohne \ oder ähnliche Sondermarkierungen. Zählervariablen werden durch die Kommandos

`\newcounter{`*Zv*`}` oder `\newcounter{`*Zv*`}[`*Zv′*`]`

deklariert, wobei *Zv* der Name der deklarierten Variablen ist und *Zv′* der einer anderen Zählervariablen. Der deklarierte Zähler wird mit dem Wert 0 initialisiert. Im zweiten Fall wird er dem Zähler *Zv′* untergeordnet, d. h. *Zv* wird automatisch auf 0 zurückgesetzt, wenn *Zv′* um 1 erhöht wird.

Die wichtigsten Kommandos zur Definition bzw. Änderung des Wertes von Zählvariablen sind:

- `\setcounter{`*Zv*`}{`*Zahl*`}`
  bindet *Zv* an die angegebene Zahl;
- `\addtocounter{`*Zv*`}{`*Zahl*`}`
  erhöht den Wert von *Zv* um die angegebene Zahl, die auch negativ sein kann;
- `\stepcounter{`*Zv*`}`
  erhöht den Wert von *Zv* um 1 und setzt alle untergeordneten Zähler auf 0.

Der aktuelle Zählerstand, d. h. die an die Zählervariable gebundene Zahl, kann durch das Kommando `\value{`*Zv*`}` verfügbar gemacht werden. Dieses Kommando kann z. B. als zweites Argument von `\setcounter` anstelle einer expliziten Zahl benutzt werden.

Der Befehl `\value{`*Zv*`}` kann jedoch *nicht* als Bestandteil von normalem Text benutzt werden. Um einen Zählerstand in Text zu verwandeln, gibt es die Befehle

`\arabic{`*Zv*`}` `\roman{`*Zv*`}` `\Roman{`*Zv*`}` `\alph{`*Zv*`}` `\Alph{`*Zv*`}`

Wenn diese Befehle in dieser Reihenfolge auf den Kapitelzähler `chapter` angewandt werden, ergibt sich

7 vii VII g G

Auf den Seitenzähler `page` sind an dieser Stelle nur die ersten drei Befehle anwendbar; für die zwei letzten ist sein Wert zu groß. Es ergibt sich

184 clxxxiv CLXXXIV

Längenvariablen

Längenvariablen können an Längen (mit und ohne Variabilität) gebunden werden. Wichtige vordefinierte Längenvariablen sind

`\textheight`	normale Texthöhe
`\textwidth`	normale Textbreite
`\linewidth`	Textbreite in aktueller Umgebung
`\baselineskip`	Minimalabstand zwischen den Grundlinien zweier aufeinanderfolgender Textzeilen in einem Paragraphen
`\parskip`	zusätzlicher Abstand vor einem Paragraphen

Im Gegensatz zu Zählern werden also die Namen von Längenvariablen durch ein besonderes Zeichen eingeleitet, nämlich \. Längenvariablen werden durch das Kommando

`\newlength{`*Lv*`}`

deklariert und mit 0 initialisiert, wobei *Lv* der Name der deklarierten Variablen ist. Die wichtigsten Kommandos zur Definition bzw. Änderung des Wertes von Längenvariablen sind:

- `\setlength{`*Lv*`}{`*Länge*`}`
  bindet *Lv* an die angegebene Länge;
- `\addtolength{`*Lv*`}{`*Länge*`}`
  erhöht den Wert von *Lv* um den angegebenen Betrag;
- `\settowidth{`*Lv*`}{`*Text*`}`
  bindet *Lv* an die Breite, die der angegebene Text nach dem Setzen hätte;
- `\settoheight{`*Lv*`}{`*Text*`}` und `\settodepth{`*Lv*`}{`*Text*`}`
  arbeiten analog mit der Höhe bzw. Tiefe.

Wo in den oben aufgezählten Befehlsformaten das Wort *Länge* vorkommt, kann eine explizite Längenangabe wie `5mm`, eine Längenvariable wie `\linewidth` oder eine Längenvariable mit Vorfaktor wie `0.5\linewidth` stehen. Es gibt noch viele andere Befehle, die solche Längenangaben erfordern; der Befehl `\hspace`{*Länge*} zum Beispiel erzeugt einen Leerraum der angegebenen Länge.

Auf den Wert einer Längenvariablen kann also direkt zugegriffen werden ohne einen speziellen Befehl wie `\value`{*Zähler*}. Ähnlich wie Zähler können Längenvariablen aber nicht direkt im Text vorkommen; ihr Wert wird erst durch das Kommando `\the` als Text verfügbar. Die oben genannten systemdefinierten Längenvariablen liefern zum Beispiel

`\the\textheight`	550.0pt
`\the\textwidth`	364.19527pt
`\the\linewidth`	364.19527pt
`\the\baselineskip`	12.0pt
`\the\parskip`	8.61108pt plus 2.15277pt minus 2.15277pt

Die Längenangabe erfolgt immer in der Einheit `pt`, auch wenn bei `\setlength` eine andere Einheit benutzt worden war. Die zweite Zahl bei `\parskip` ist die Dehnbarkeit und die dritte die Schrumpfbarkeit; die Werte der anderen vier Variablen sind starr, d. h. nicht dehn- oder schrumpfbar.

Makronamen als Textvariablen

Makroersetzung ist ein sehr allgemeines Konzept, das wir in Abschnitt 7.3 genauer besprechen. Hier betrachten wir nur den einfachsten Fall, wo Makronamen als Textvariablen benutzt werden, also an ein Stück Text gebunden werden. Mit Text ist hier ein Stück LaTeX-Dokumentbeschreibung gemeint, das unter anderem Positionierungs- und Fontwechselbefehle enthalten kann. Die beiden Logos TeX und LaTeX werden z. B. durch die vordefinierten Textvariablen `\TeX` und `\LaTeX` erzeugt.

Wie man sieht, bestehen die Namen von Textvariablen aus einer Folge von Buchstaben, die von dem Zeichen \ eingeleitet wird. (Es gibt auch noch Namen anderer Struktur, die wir hier aber nicht betrachten wollen.) Im Gegensatz zu Zähler- und Längenvariablen müssen Textvariablen nicht deklariert werden; sie werden direkt an einen Wert gebunden. Dazu gibt es in LaTeX drei verschiedene Formate:

`\newcommand`{*Tv*}{*Text*}
`\renewcommand`{*Tv*}{*Text*}
`\providecommand`{*Tv*}{*Text*}

Alle drei Befehle haben im wesentlichen das Ziel, die Textvariable *Tv* an den Wert *Text* zu binden. Sie unterscheiden sich in ihrer Anwendbarkeit: `\newcommand` ist nur anwendbar, wenn *Tv* noch nicht definiert ist; `\renewcommand` dagegen nur,

wenn *Tv* bereits definiert ist; die alte Definition wird überschrieben. Die Form `\providecommand`, die es erst in neueren Versionen von LaTeX gibt, ist immer anwendbar, hat aber nur dann eine Wirkung, wenn *Tv* noch nicht definiert war. Zusätzlich gibt es aus dem unterliegenden TeX-System das Format

`\def` *Tv* {*Text*}

(ohne geschweifte Klammern um *Tv*!), das immer anwendbar ist und immer wirkt. Alle diese Befehlsformate sind eigentlich nur Spezialfälle von allgemeineren, die wir in Abschnitt 7.3.4 kennenlernen werden.

Textvariablen können (fast) überall in der Dokumentenbeschreibung benutzt werden, indem einfach ihr Name hingeschrieben wird. Er wird dann durch den an die Variable gebundenen Text ersetzt. Dabei ist eine kleine Schwierigkeit zu beachten: während der Anfang des Namens einer Textvariablen durch das Zeichen \ leicht erkennbar ist, ist es mit dem Ende schwieriger bestellt. Die Festlegung ist, daß der Name beim ersten Zeichen endet, das kein Buchstabe ist. Die Eingabe `\TeX?` erzeugt also z. B. die Ausgabe „TeX?“. Um dagegen einen Text wie „the TeXbook“ zu erzeugen, kann nicht `the \TeXbook` geschrieben werden, da TeX dann nach der Variablen `\TeXbook` suchen würde, die höchstwahrscheinlich undefiniert ist. Es gibt vielmehr die Regel, daß Leerzeichen nach einer Textvariablen verschluckt werden; die korrekte Eingabe ist also `the \TeX book`. Wie kann dann aber der Text „TeX oder LaTeX?“ erzeugt werden? Die einfachste Möglichkeit ist, das Leerzeichen nach TeX mit \ zu schützen: `\TeX\ oder \LaTeX?`.

Vergleich von Zähler- und Textvariablen

Äußerlich sehen Zähler- und Textvariablen gleich aus; ihre Namen sind Buchstabenfolgen, die von dem Zeichen \ eingeleitet werden. Obwohl wir die Handhabung dieser beiden Arten von Variablen oben genau beschrieben haben, lohnt sich ein eingehender Vergleich.

Angenommen, wir deklarieren eine Beispiel-Zählervariable `\bspZ` und binden sie an einen Wert von 3 mm:

```
\newlength{\bspZ}       \setlength{\bspZ}{3mm}
```

Es hindert uns jetzt keiner, eine Beispiel-Textvariable `\bspT` zu definieren und an den *Text* „3mm“ zu binden:

```
\newcommand{\bspT}{3mm}
```

Als Textvariable kann `\bspT` natürlich in normalem Text benutzt werden; sie bewirkt die Ausgabe des Textes `3mm`. Die Zählervariable `\bspZ` kann dagegen in normalem Text nicht direkt benutzt werden; die Eingabe von `\the\bspZ` erzeugt den Text `8.53581pt` und nicht etwa `3mm`, da Längen immer in der Einheit `pt` ausgegeben werden.

An den Stellen, wo LaTeX eine Länge erwartet, sind dagegen beide Variablen ohne Unterschied benutzbar; sowohl `\hspace{\bspZ}` als auch `\hspace{\bspT}` sind erlaubt und äquivalent zu `\hspace{3mm}`. Die Sache sieht aber anders aus, wenn Vorfaktoren benutzt werden: `\hspace{2\bspZ}` bedeutet `\hspace{6mm}`, während `\hspace{2\bspT}` zu `\hspace{23mm}` wird, da der Text `3mm` direkt hinter die `2` gesetzt wird, was den Text `23mm` liefert.

7.2.2 Das Binden von Variablen in interaktiven Systemen

Interaktive Systeme haben meist andere Mechanismen als Batch-Systeme, um Variablen einzuführen und an Werte zu binden. Sowohl das Kreieren als auch das Binden als auch das Benutzen von Variablen erfolgen über Menüs.

Variablen des *FrameMaker* werden in Menüfenstern angezeigt. Der Benutzer kann auch neue Variablen kreieren, die dann im Menüfenster erscheinen. Sie können darin durch Mausklick aktiviert werden. Anschließend kann der Wert der aktivierten Variablen verändert werden oder an einer anzugebenden Stelle in das Dokument eingefügt werden. Da der *FrameMaker* das Dokument nach einer solchen Einfügung wieder neu darstellt, steht an der Stelle der Einfügung der Wert, an den die Variable gebunden wurde. Allerdings ist weiterhin diese Einfügung als angewandtes Vorkommen einer Variablen gespeichert. Spätere Änderungen des Wertes der Variablen wirken sich deshalb auf alle solche angewandten Vorkommen aus. Eine Edieroperation an dem Wert der Variablen bei einem ihrer angewandten Vorkommen ist möglich. Allerdings geht dabei verloren, daß dort einmal ein angewandtes Vorkommen der Variablen stand.

Einige wichtige vordefinierte Variablen des *FrameMaker* sind `Current Page #`, die Nummer der aktuellen Seite, `Page Count`, die Gesamtzahl der Seiten, und `Running H/F 1`, das ist der Titel des Dokuments als laufende Kopf- und Fußzeile.

7.3 Makros

Makros sind ein Abkürzungsmechanismus für Texte. Eine Makrodefinition führt einen Namen für einen Ersetzungstext ein, der im allgemeinen weitere Makrodefinitionen, Makrobenutzungen und auch variable Bestandteile enthalten kann. Eine einfache Form davon haben wir oben schon bei der Besprechung von Variablen in LaTeX kennengelernt. Fassen wir noch einmal kurz zusammen, was wir dort gelernt haben, wobei wir von der konkreten Erscheinungsform von LaTeX abstrahieren.

In einem Text, der durch ein Formatierprogramm läuft, können vom Autor Makronamen durch eine Deklaration eingeführt und durch eine eventuell damit kombinierte Definition an einen Wert gebunden werden. Dafür gibt es ein bestimmtes syntaktisches Format, z. B.

```
x = "hier steht ein x".
```

Eine solche Deklaration führt ein Makro namens `x` ein und bindet es an den Wert `hier steht ein x`. Soll der Wert von `x` an einer Stelle im Dokument benutzt werden, so schreibt man `x` an diese Stelle hin. Solch ein Vorkommen heißt angewandtes Vorkommen von `x`. Der Formatierer ersetzt `x` dann durch `hier steht ein x`. Eventuell ist `x` schon früher einmal definiert worden. Dann überschreibt die neue Definition im allgemeinen den alten Wert von `x` mit dem in der Neudefinition stehenden Wert. Damit angewandte Vorkommen von Makros als solche erkannt werden, müssen sie eine erkennbare Syntax haben. Sonst würde z. B. `x` auch an unerwünschten Stellen, etwa als Teil von `Text`, als Makroname interpretiert und dort ebenfalls durch `hier steht ein x` ersetzt. Das ergäbe dann den wenig sinnvollen Text `Tehier steht ein xt`.

7.3.1 Ein Beispiel-Makrosystem

Im folgenden führen wir ein Makrosystem ein, das nichts mit TEX oder LATEX zu tun hat. Es soll zur Veranschaulichung einiger Konzepte dienen und baut auf dem System GPM (general purpose macroprocessor) auf, das von Christopher Strachey realisiert wurde [Str65].

In der Definition %define (mname, text) wird der Name mname für den Ersetzungstext text eingeführt. Tritt im Dokument anschließend %mname auf, wir reden von einem *Makroaufruf*, so wird an dieser Stelle dieser Aufruf durch text ersetzt. Man sagt, das Makro wird *expandiert*. Dieser Ersetzungsprozeß wird unten noch genauer geschildert.

Makros können auch *Parameter* haben. Dies ermöglicht es, bei verschiedenen Aufrufen verschiedene Texte einzusetzen. Hier ist ein Beispiel: Sei der Makroname abc für den Ersetzungstext ab$1c$2ab eingeführt worden. Die sogenannten *formalen Parameter* $1 und $2 im Ersetzungstext werden bei dem Aufruf %abc(xy,pq) z. B. durch die *aktuellen Parameter* xy und pq ersetzt; daher ergibt der Aufruf den Text abxycpqab.

Allgemein gibt es die Bezeichnungen $1, $2, ... $9 für bis zu neun formale Parameter, die beim Aufruf durch den ersten bis neunten aktuellen Parameter ersetzt werden. Beim Aufruf %mname(a,b,c) werden z. B. a, b und c für $1, $2 bzw. $3 eingesetzt. In einem Ersetzungstext kann derselbe formale Parameter mehrfach auftreten oder ganz fehlen.

Es gibt keine Einschränkungen dafür, wo Makroaufrufe vorkommen dürfen, z. B. können sie auch in aktuellen Parametern, im Ersetzungstext und selbst im Makronamen in einer Definition auftreten. Nehmen wir zum Beispiel an, daß die Zuordnung von Makronamen zu Ersetzungstexten wie in Tabelle 7.1 definiert wurde. Dann ergeben sich die Makroexpansionen aus Tabelle 7.2. Schauen wir uns die etwas komplizierteren Fälle genauer an. Im dritten Beispiel ist der Aufruf %a(c) aktueller Parameter von a. Sein Wert ist aca. Wird dieser in den Ersetzungstext von a eingesetzt, so ergibt sich aacaa. Das Makro b enthält den Aufruf %a(x$1x) von a in seinem Ersetzungstext. Dabei bezeichnet $1 den ersten formalen Parameter von b.

Makroname	Ersetzungstext
a	a$1a
b	b%a(x$1x)b
apa	p$1$1p

Tabelle 7.1: Zuordnung von Makronamen zu Ersetzungstexten

Makroaufruf	Ergebnis
%a(c)	aca
%a(aca)	aacaa
%a(%a(c))	aacaa
%a(xdx)	axdxa

Makroaufruf	Ergebnis
%b(d)	baxdxab
%a(p)	apa
%apa(y)	pyyp
%%a(p)(y)	pyyp

Tabelle 7.2: Einige Beispielexpansionen

Wird b mit dem aktuellen Parameter d aufgerufen, so wird also a mit dem aktuellen Parameter xdx aufgerufen und produziert als Ergebnis axdxa, so daß der Aufruf %b(d) als Ergebnis baxdxab erzeugt. Der letzte Aufruf %%a(p)(y) schließlich enthält auf der Position des Namens den Aufruf %a(p). Dieser Aufruf hat als Ergebnis apa. Somit ist apa der Makroname im äußeren Aufruf. Seine Anwendung auf y ergibt damit pyyp.

Es gibt die Möglichkeit, die Expansion von Makronamen in einem Text zu verhindern. Dazu wird der betreffende Text in eckige Klammern eingeschlossen. Dann wird kein Makroaufruf innerhalb der Klammern expandiert. Allerdings wird ein Paar von öffnenden und schließenden eckigen Klammern entfernt. Mit den Makros aus Tabelle 7.1 ergeben sich die Resultate in Tabelle 7.3.

Klammern können auch geschachtelt auftreten, wie z. B. in [a[b[c]a]]. Die Zuordnung von öffnenden und schließenden eckigen Klammern in korrekt geklammertem Text ist aber eindeutig, weil Paare von zueinander gehörenden öffnenden und schließenden Klammern nicht überlappend auftreten können. Entweder ist ein Paar vollständig von einem anderen Paar umgeben, oder die beiden Paare klammern disjunkte Stücke Text ein. Es wird immer das äußerste Paar Klammern entfernt.

Unser Beispiel-Makrosystem ähnelt dem UNIX-Makrosystem M4 [KR77]. Ein Unterschied ist, daß man in M4 kein Fluchtsymbol % vor Makronamen schreibt. M4 expandiert Vorkommen von Makronamen, wo immer es sie identifizieren kann. Das ist dann der Fall, wenn diese Makronamen, die aus alphanumerischen Zeichen aufgebaut sind, durch nicht-alphanumerische Zeichen begrenzt sind. Ein Vorkommen eines Makronamens als echtes Teilwort eines alphanumerischen Textes wird

Eingabetext	Ergebnis
q[%a(c)]r	q%a(c)r
%a([%a(x)])	a%a(x)a
%b([%a(x)])	bax%a(x)xa
q[[%a(c)]]r	q[%a(c)]r

Tabelle 7.3: Unterdrückung von Makroexpansion

also dort nicht expandiert. Außerdem verwendet M4 zum Schützen statt eckiger Klammern öffnende und schließende Apostrophs. M4 baut auf dem oben erwähnten, von Christopher Strachey realisierten Makrosystem GPM (general purpose macroprocessor) auf.

7.3.2 Makroexpansion

Der Prozeß der Makroexpansion läuft folgendermaßen ab. Der Makroexpander bekommt einen Eingabetext und produziert einen Ausgabetext. Er liest den Eingabetext als einen Eingabestrom von Zeichen von links nach rechts und kopiert die Zeichen aus dem Eingabestrom in den Ausgabestrom. Nur wenn er eine Makrodefinition oder einen Makroaufruf antrifft, unterbricht er diese Normaltätigkeit und verarbeitet die Definition bzw. den Aufruf, wie gleich beschrieben.

Makrodefinition

Eine Makrodefinition assoziiert einen Makronamen mit einem Ersetzungstext. Alle zu einem Verarbeitungszeitpunkt gültigen Makronamen sind zusammen mit ihren Ersetzungstexten in einer Liste, der sogenannten *Umgebung*, gespeichert. Wird eine Makrodefinition verarbeitet, so wird dabei ein Paar, bestehend aus Makroname und Ersetzungstext, erzeugt und vorne an die Umgebung angefügt. Da die Umgebung von vorne nach hinten durchsucht wird, werden also immer die zuletzt eingefügten Definitionen zuerst gefunden. Makrodefinitionen können also durch neue Definitionen überschrieben werden.

Beispiel: Die Makros a, b und apa aus Tabelle 7.1 werden durch die folgenden Definitionen eingeführt:

```
%define(a,[a$1a])        %define(b,[b%a(x$1x)b])        %define(apa,[p$1$1p])
```

Es empfiehlt sich, den Ersetzungstext in einer Makrodefinition durch eckige Klammern zu schützen, denn in mancher Beziehung verhält sich define wie ein Makro. Es würde nämlich z. B. formale Parameter $1, $2 usw., die im Ersetzungstext des definierten Makros auftreten, als seine eigenen formalen Parameter ansehen. Außerdem würde es versuchen, alle Makroaufrufe im Ersetzungstext zu expandieren.

Makrodefinitionen können in Makrodefinitionen stehen, und zwar sowohl in Makronamen wie in Ersetzungstexten. Wenn ein Aufruf verarbeitet ist, wird die Liste der aktuellen Parameter gelöscht. Alle Makros, deren Definitionen in dieser Liste stehen, werden ebenfalls wieder (aus der Umgebung) gelöscht. Sie sind anschließend nicht mehr bekannt. Deshalb heißen solche Makros *temporär*. Betrachten wir folgende Situation: Es gebe eine gültige Definition von a. Nun wird eine Definition von b verarbeitet, die in ihrem Ersetzungstext eine Definition von a enthält. Dann gilt diese neue Definition von a nur ab ihrer Verarbeitung bis zum Ende der Verarbeitung der Definition von b. Anschließend gilt wieder die alte Definition von a. Zwischendurch gab es also ein temporäres Makro a.

Ein Beispiel: Der Aufruf %a(x,u%define(a,[$1$2$1])) benutzt eine lokale Definition von a, wendet diese auf x und u an und erzeugt als Ergebnis xux.

Makroaufruf

Der Makroexpander unterbricht das Kopieren vom Eingabestrom in den Ausgabestrom, wenn er einen Makroaufruf antrifft. Dann führt er die folgenden Schritte in der angegebenen Reihenfolge aus.

1. Der Makroname und die aktuellen Parameter werden von links nach rechts ausgewertet. Darin auftretende Makroaufrufe werden ihrerseits expandiert. Ein Aufruf von define z. B. führt zur Erweiterung der Umgebung um die entsprechenden Makrodefinitionen.
2. Ist die Liste aus Makroname und aktuellen Parametern fertig, so wird die aktuelle Umgebung nach einer Definition des Makronamens durchsucht, und zwar von vorn nach hinten, so daß die letzte hinzugefügte Definition als erste gefunden und dann benutzt wird. Gibt es keine Definition in der Umgebung, so wird ein Fehler gemeldet.
3. Jetzt wird der Ersetzungstext von links nach rechts verarbeitet. Dabei werden Vorkommen der formalen Parameter $1, $2 usw. durch die ausgewerteten aktuellen Parameter ersetzt. Die aktuellen Parameter erfahren dabei keine weitere Auswertung.
4. Wird das Ende des Ersetzungstextes erreicht, so wird die Liste der aktuellen Parameter gelöscht. Alle bei ihrer Verarbeitung definierten Makros werden aus der Umgebung gelöscht.

Anschließend wird mit der Verarbeitung des Eingabestroms nach dem Aufruf fortgefahren.

7.3.3 Call-by-value und call-by-name, statische und dynamische Bindung

Man spricht von Makro*aufrufen* in Analogie zu Prozeduraufrufen in Programmen. Prozeduren werden allerdings aufgerufen, wenn ein Programm ausgeführt wird.

Es handelt sich hierbei nicht um eine textuelle Ersetzung. Prozeduren haben im allgemeinen auch Parameter. Die Semantik der Programmiersprache legt fest, wie die aktuellen Parameter an die aufgerufene Prozedur übergeben werden. Man redet von *call-by-value*-Übergabe, wenn die aktuellen Parameter erst ausgewertet und dann die Ergebnisse an die Prozedur übergeben werden. Dagegen spricht man von *call-by-name*-Übergabe, wenn die aktuellen Parameter unausgewertet an die Prozedur übergeben werden. Die Auswertung geschieht in diesem Fall erst, wenn die Prozedur die Werte benötigt; das ist dann der Fall, wenn sie auf die korrespondierenden formalen Parameter zugreift.

Betrachten wir erneut, wie aktuelle Parameter von Makroaufrufen behandelt werden. Sie werden erst ausgewertet, dann wird das Ergebnis in den Ersetzungstext einkopiert, was einer call-by-value-Übergabe der aktuellen Parameter entspricht. Natürlich wäre auch eine call-by-name-Übergabe möglich. Dann würden die aktuellen Parameter unausgewertet in den Ersetzungstext einkopiert. Erst wenn der Ersetzungstext verarbeitet wird, werden dann eventuell in den aktuellen Parametern stehende Makroaufrufe expandiert. Wir werden im nächsten Abschnitt sehen, daß das Makrosystem von TEX die call-by-name-Übergabe der Parameter benutzt. Man kann denselben Effekt auch erreichen, wenn man die aktuellen Parameter in eckige Klammern einschließt.

Ein weiteres wichtiges Konzept aus Programmiersprachen sind die *Bindungsregeln*. Den Namen „Bindung“ haben wir bisher schon benutzt, aber in einem etwas anderen Sinne, nämlich als Bindung einer Variablen an einen Wert. Bindungsregeln klären, worauf sich ein angewandtes Variablenvorkommen bezieht, wenn es darüber Zweifel geben kann. In Programmen wie auch in Texten, die Makros definieren und benutzen, gibt es *definierende* und *angewandte* Vorkommen von Namen. Klar, eine Makrodefinition definiert einen Namen, und ein Makroaufruf wendet ihn an. Wenn ein Text mehrere Definitionen eines Makros enthalten kann, so stellt sich die Frage, zu welchem definierenden Vorkommen eines Namens ein angewandtes Vorkommen gehört. Zwei verschiedene Regeln sind gebräuchlich, statische und dynamische Bindung. Die *dynamische Bindung* ordnet einem angewandten Vorkommen das letzte während der Ausführung bzw. der Makroexpansion vorher angetroffene definierende Vorkommen zu. *Statische Bindung* dagegen benutzt die Struktur des Texts, um die Zuordnung von angewandten Vorkommen zu definierenden Vorkommen durchzuführen. Dazu gibt es im Text geschachtelte Klammerstrukturen, in Programmen etwa Blöcke und Prozeduren, in Texten etwa Kapitel, Abschnitte und Listen, welche die *Sichtbarkeit* von Definitionen begrenzen; nennen wir sie *Sichtbarkeitsbereiche*. Eine Definition in einem inneren Sichtbarkeitsbereich überdeckt für diesen Bereich eventuell außen vorhandene Definitionen.

An folgendem Beispiel kann man den Unterschied zwischen statischer und dynamischer Bindung sehen. Wir nehmen an, daß in unserer Dokumentsorte Gültigkeitsbereiche für Definitionen existieren. Ihr Anfang werde durch `%begin` und ihr Ende durch `%end` gekennzeichnet. Außerdem nehmen wir an, daß unser Makrosystem zur Übergabe der Parameter den call-by-value-Mechanismus benutzt.

```
%begin    %define(a,[aussen])
          %define(b,[$1 a kommt von %a])    %b(1.)
          %begin    %define(a,[innen])    %b(2.)    %end
          %b(3.)
%end
```

Betrachten wir jetzt die Ergebnisse der drei Aufrufe von b unter statischer und unter dynamischer Bindung. Das Ergebnis des Aufrufs %b(1.) ist in beiden Fällen 1. a kommt von aussen. Der Aufruf %b(2.) ergibt bei statischer Bindung 2. a kommt von aussen, weil die „äußere“ Definition von a für b sichtbar ist. Bei dynamischer Bindung ergibt er 2. a kommt von innen, weil die „innere“ Definition von a die letzte verarbeitete ist. Der Aufruf %b(3.) dagegen liefert wieder in beiden Fällen 3. a kommt von aussen, da er sich außerhalb des Gültigkeitsbereichs der „inneren“ Definition von a befindet.

Wie wir oben gesehen haben, überschreibt eine neue Definition eines Makros eine eventuell schon vorhandene. Somit realisiert unser Makrosystem die dynamische Bindung. Dasselbe gilt für das weiter unten besprochene Makrosystem von TEX.

7.3.4 Das Makrosystem von TEX

Ein für die Dokumentenverarbeitung wichtiges Makrosystem ist das von TEX. Dieses wird im folgenden näher, wenn auch nicht erschöpfend, behandelt. Fangen wir damit an, wie sich die Syntax von der unseres Beispielsystems unterscheidet. Ein Makro, in TEX-Terminologie eine *Kontrollsequenz*, wird durch eine `\def`-Anweisung definiert. Sie hat folgende Struktur:

`\def` *Name Parameterteil* {*Ersetzungstext*}

Hierdurch wird das Makro mit dem Namen *Name* eingeführt. Der Ersetzungstext wird in geschweifte Klammern eingeschlossen, die nicht Teil des Ersetzungstexts sind; beim Ersetzen verschwinden sie. Wie bei uns darf ein Makro in TEX bis zu 9 Parameter haben. Die formalen Parameter werden im Ersetzungstext mit #1, #2 usw. bezeichnet. Dazwischen steht ein Parameterteil, wo die formalen Parameter schon einmal genannt werden müssen. Im einfachsten Fall werden sie einfach hintereinander aufgezählt. In Aufrufen werden dann die aktuellen Parameter einzeln in geschweifte Klammern eingeschlossen. Wie beim Ersetzungstext sind diese Klammern kein Teil der Argumente; sie fallen weg, wenn die Argumente in den Ersetzungstext einkopiert werden. In TEX wird jeder Makroname mit dem Zeichen \ eingeleitet, das als Bestandteil des Namens aufgefaßt wird. Damit ergeben sich die folgenden Korrespondenzen zwischen unserem System und TEX:

%define(m,[abc])	`\def\m{abc}`
%define(abcd,[a$1b$2c$1d])	`\def\abcd#1#2{a#1b#2c#1d}`
%abcd(xy,pq)	`\abcd{xy}{pq}`
%define(b,[b%a(x$1x)b])	`\def\b#1{b\a{x#1x}b}`

Expansionsstrategie

Unser Beispiel-Makrosystem ist in einem gewissen Sinne „eifrig“, weil es nämlich immer versucht, Makroaufrufe zu expandieren. Man kann es allerdings durch Einschließen in eckige Klammern daran hindern. Diese call-by-value-Regel für die Expansion heißt tatsächlich bei den funktionalen Programmierern *eager evaluation*. Das Makrosystem von TEX ist im Gegensatz dazu eher „faul“, weil es Makroaufrufe zu einem möglichst späten Zeitpunkt expandiert. Aktuelle Parameter eines Makroaufrufs werden also ohne vorherige Expansion in den Ersetzungstext einkopiert, und bei einer Makrodefinition wird der Ersetzungstext ohne Expansion an den Makronamen gebunden. Dies entspricht der oben geschilderten call-by-name-Regel. Eine ähnliche Auswertungsregel heißt bei den funktionalen Programmierern deshalb auch *lazy evaluation*.

TEX bietet allerdings einige Möglichkeiten an, diese Expansionsstrategie zu beeinflussen. Wenn z. B. `\edef` statt `\def` benutzt wird, dann wird der Ersetzungstext schon expandiert, bevor er an den Makronamen gebunden wird. Andererseits kann man auch die Expansion von Makroaufrufen verhindern, wo die Expansionsregeln von TEX sie vorschreiben würden. Man schreibt dazu `\noexpand` vor den Aufruf.

Gültigkeitsbereiche und Art der Bindung

Die Gültigkeit einer TEX-Makrodefinition ist auf die Gruppe begrenzt, in der die Definition steht. Sie beginnt mit der Definition und endet mit dem Ende der Gruppe oder einer neuen Definition für denselben Makronamen. Für die Zuordnung von angewandten zu definierenden Vorkommen von Namen wird die dynamische Bindung benutzt. Das Beispiel am Ende von Abschnitt 7.3.3 lautet in TEX:

```
{ \def\a{au"sen}  \def\b#1{#1 a kommt von \a}  \b{1.}\\
  { \def\a{innen} \b{2.}\\ }
  \b{3.}
}
```

wobei die Gruppen durch die freistehenden geschweiften Klammern gebildet werden und das Kommando \\ einen Zeilenwechsel bewirkt. Wenn das gerade angegebene TEX-Programmstück ausgeführt wird, ergibt sich

1. a kommt von außen
2. a kommt von innen
3. a kommt von außen

was die dynamische Bindung belegt. Statische Bindung kann erzwungen werden, wenn bei der Definition von `\b` `\edef` (expanded definition) statt `\def` benutzt wird. Dann steht nämlich statt `\a` schon `au"sen` im Ersetzungstext von `\b` und es ergibt sich

1. a kommt von außen
2. a kommt von außen
3. a kommt von außen

Muster im Parameterteil

Der Parameterteil in den Makrodefinitionen von TEX weist gegenüber den meisten anderen Makrosystemen eine Besonderheit auf. Man kann nämlich die Anwendbarkeit von Makros dadurch einschränken, daß man die aktuellen Parameter im Aufruf in einen gewissen Kontext einbettet. Zum Beispiel verlangt die Definition

```
\def\hh<#1?#2:#3>{(#3)(#2)(#1)}
```

daß die aktuellen Parameter von `<` und `>` eingefaßt und durch `?` sowie `:` getrennt werden. Ein Aufruf `\hh<ab?cd:ef>` ergibt dann den Text `(ef)(cd)(ab)`. Wird dieses Makro aufgerufen, ohne daß die aktuellen Parameter so, wie es die Definition vorschreibt, eingefaßt bzw. getrennt werden, so meldet TEX einen Fehler.

Makros in LATEX

In LATEX ist das Makrosystem von TEX immer noch vorhanden und benutzbar. Außerdem gibt es drei zusätzliche Befehle `\newcommand`, `\renewcommand` und `\providecommand` zur Definition von Makros. Die Unterschiede der drei Befehle bezüglich Erst- und Umdefinitionen wurden bereits in Abschnitt 7.2.1 besprochen. Hier wollen wir erläutern, wie man mit diesen Befehlen Makros mit Parametern definieren kann. Die Methode ist bei allen drei Befehlen dieselbe; wir konzentrieren uns daher auf `\newcommand`.

Die einfachste Form dieser Befehle haben wir bereits in Abschnitt 7.2.1 kennengelernt. Durch

`\newcommand`{*Makroname*}{*Ersetzungstext*}

wird der Makroname an den gegebenen Ersetzungstext gebunden, der keine formalen Parameter enthalten darf. Makros mit n Parametern, $1 \leq n \leq 9$, werden durch

`\newcommand`{*Makroname*}[n]{*Ersetzungstext*}

definiert. Wie in TEX werden die formalen Parameter im Ersetzungstext als `#1` bis `#9` geschrieben. Der Aufruf erfolgt wie in TEX als

Makroname{Arg_1}...{Arg_n}

In neueren LATEX-Versionen gibt es eine weitere interessante Form der Makrodefinition. Durch

`\newcommand`{*Makroname*}[*n*][*Text*$_1$]{*Ersetzungstext*}

wird ein Makro definiert, dessen erster Parameter optional ist. Es hat zwei verschiedene Aufrufformate:

Makroname[*Arg*$_1$]{*Arg*$_2$}...{*Arg*$_n$} und *Makroname*{*Arg*$_2$}...{*Arg*$_n$}

Im ersten Fall wird der formale Parameter `#1` durch *Arg*$_1$ ersetzt und im zweiten Fall durch den *Text*$_1$ aus der Makrodefinition. Nach der Definition

```
\newcommand{\Bsp}[2][sehr]{Es ist #1 #2}
```

ergibt also der Aufruf `\Bsp{fr"uh}` den Text „Es ist sehr früh“, während der Aufruf `\Bsp[viel zu]{sp"at}` zu „Es ist viel zu spät“ expandiert.