Universität Rostock

Traditio et Innovatio

Automatische Sprachübersetzung von LaTeX-Dokumenten

Name:	Hendrik Theede
Matrikelnummer:	221201256
Abgabedatum:	02.12.2025
Betreuer und Gutachter:	Prof. Dr. rer. nat. habil. Clemens H. Cap Universität Rostock Fakultät für Elektrotechnik und Informatik

Bachelorarbeit am Lehrstuhl für Informations- und Kommunikationsdienste

Fakultät für Elektrotechnik und Informatik

Abstrakt

placeholder

Inhaltsverzeichnis

1 Einleitung

1.1 Hintergrund

Herkömmliche Software zur Übersetzung von menschlicher Sprache auf TEX-Quellcode anzuwenden, erzeugt schnell Dokumente, welche entweder nicht vollständig übersetzt wurden oder sich nicht mehr kompilieren lassen. Mit Hilfe von Google Translate lassen sich wesentliche Gründe hierfür finden und wie sich diese äußern. Beispielsweise führt eine Übersetzung von `hello wor\textit{ld}` nicht zu `Hallo We\textit{lt}`, sondern zu `hallo wor\textit{ld}`. Abgesehen von der Frage, wo die kursive Hervorhebung im eigentlichen String erfolgen soll, werden Leser eines kompilierten Dokumentes das Wort "Welt" erkennen. Zuvor beschriebene Zeichenkette wird von TEX zu "hello wor*ld*" aufgelöst, in welcher das Wort "world" für einen menschlichen Leser als das englische Wort für "Welt" erkenntlich bleibt. Fehlt die Kenntnis über eine der Sprachen (DE,EN), würde einem monolingualen Leser Teil der Wortkette geraubt werden. Selbstverständlich sind die Wörter "world" und "Welt" einander sehr nahe und auch eine Formulierung der Art "Hallo Welt" lässt Vermutungen gegenüber eines größeren Kontexts zu.Anders wäre dies, wenn das Auslassen von auch nur einem Wort keine Rückschlüsse mehr auf einen größeren Kontext mehr zulässt. `$\mathbb{P}$robability density function` wäre ein denkbarer stilistischer Weg bereits in z.B. einem Folientitel bereits eine Notation für eine Wahrscheinlichkeitsdichtefunktion einzuführen. Hierbei würde der Verlust des Wortes "probability" den stochastischen Kontext aufheben. Der Verlust des Wortes "density" würde einen Kontext innerhalb der Stochastik verändern und ohne das Wort "function" ist fraglich, wovon die Rede ist. Vor allem in größeren Dokumenten könnten hierdurch Logikbrüche entstehen.

Eine Betrachtung eines "Übersetzers" als Konzept veranschaulicht die Problematik auf abstrakterer Ebene. Sollte der Kontext des Dokumentes unbekannt sein, werden sich unausweichlich semantische Fehler einschleichen. Bereits das gezeigte Beispiel könnte für z.B. eine Folie einer Lesung den restlichen Kontext der Seite entfernen und dadurch die Möglichkeit bieten umgangssprachliche Bedeutungen in Wörter zu interpretieren, anstatt einer Mathematischen (bspw. "ungerade" könnte im Englischen "crooked", statt "odd" produzieren). Noch weitere sprachliche Beispiele finden sich schnellig durch Wörter mit zeitlichem/räumlichen Bezug. Der Satz `Morgen wird es regnen.` könnte ohne das Wort "morgen" als Frage mit unzureichend eingehaltener deutscher Grammatik interpretiert werden. (*Wird es regnen?*). Hierbei verliert man eine getroffene Aussage über das Wetter, welches bekanntlicherweise nur schwer vorhergesagt werden kann.

Genauso wie das Fehlen einzelner Wörter die sprachliche Bedeutung für einen Menschen brechen kann, treten ähnliche Probleme auch in TEX auf. Einzelne LATEX Makros *nicht* zu übersetzen ist unbedeutend, da sie ihre Bedeutung für einen TEX Compiler behalten. Alleine einzelne Wörter eines Makros zu übersetzen kann dazu führen, dass größere Inhalte (im Sinne: Menge an Worten) nicht mehr in einem kompilierten Dokument vorzufinden sind, was auf eine Fähigkeit von TEX zurückführbar ist. Die Möglichkeit in bestimmten Fällen eine Dateiendung auszulassen, führt beim Einbinden von anderen `.tex` Dateien in einem TEX Dokument zu fehlerhaften/fehlenden Ressourcenangaben. `\include{clock}` zu `\include{Uhr}` zu übersetzen (wie bspw. Google Translate am 06.10.2025) würde nun nicht mehr zu `\include{clock.tex}` aufgelöst werden, sondern zu `\include{Uhr.tex}` (bei welchem nicht davon auszugehen ist, dass diese Datei zur Kompilierzeit im System zwingend vorliegt).

Daher muss nach einer Lösung gesucht werden, welche diese technischen und sprachlichen Hürden überwinden kann. Neben solchen rein technischen Details, darf eine menschliche Perspektive nicht missachtet bleiben und so dürfen keine Übersetzungsprozesse dazu führen, dass in einem Dokument versteckte (und dadurch nicht lesbare) Inhalte entstehen. Solche sind zunächst aus verschiedenen Layouting-Problemen herleitbar (z.B. Skalierung von Schriftgrößen auf Webseiten, siehe Anhang A.1) und abhängig von einzelnen Sprachen unterschiedlich (da Wörter einzelner Sprachen eine minimale Pixelfläche benötigen, um angezeigt zu werden).

2 Problemfälle

Eine einzige, feste TEX-Syntax existiert theoretisch gesehen nicht, wie ein späterer Paragraph aufzeigen wird.Die Fähigkeit jegliche erdenkliche Zeichenkette (gegeben: diese ist auf einem Rechner darstellbar, siehe: Unicode Consortium (2025)) sorgt zunächst für eine unendliche Menge an testbaren Problemen. Da es unmöglich ist eine unendliche Menge an Testfällen abzudecken, herrscht zunächst eine Begrenzung auf die reine LATEX (bzw. TEX Syntax nach Knuth (1986)) und die bereits rein innerhalb dieser schnellig auffallenden Fehler, welche durch fälschlich übersetzte Zeichenketten entstehen könnten.
Für spätere Testzwecke wurden die einzelnen Problemfälle in einzelne Kategorien eingeteilt, welche nach der Häufigkeit begründet sind, mit welcher sie in einem Dokument auftreten könnten. Hierbei sind simple Probleme auf solche bezogen, welche in der reinen Erstellung und Verfassung von Dokumenten entstehen könnten und komplexe Probleme konzentrieren sich darauf, dass gezielter LATEX und TEX Befehle und Pakete genutzt werden, um durch diese weniger offensichtliche syntaktische Fehlerquellen zu erproben. Bereits diese Arten an Fehlerquellen zeigen Fälle auf, in welchen Regelbrüche gegenüber der LATEX Syntax ein Kompilieren eines Dokumentes verhindern würden. Spezielle Probleme befassen sich danach mit gezielt dahingegen erarbeitete Testfälle, welche für manuell erstellte Dokumente eher unüblich wären, technisch gesehen jedoch möglich sind und der Betrachtung nicht entzogen werden können. Weitere Schwierigkeiten werden dahingegen aufgezeigt, welche sprachlichen Hürden bei der menschensprachlichen Übersetzung aufzeigen könnten. Da diese Betrachtung nicht im Vordergrund stehen soll, jedoch für spätere Tests interessant werden könnte, werden diese vorerst an das Ende dieses Abschnittes gestellt.

2.1 Simple Probleme

Wörter werden durch Zeichenketten dargestellt, welche ihrerseits Buchstaben (bspw. lateinische, griechische, kyrillische Alphabete), Glyphen/Bilder (bspw. antikes ägyptisch, chinesisch/Hanzi, japanisch/kanji) beinhalten. Jedoch gehen die meisten Übersetzungs-Tools nicht davon aus, dass solche Zeichenketten Zeichen beinhalten, welche das folgende Wort zu einem Befehl (für eine Programmiersprache) machen. Zwar würde z.B. Google Translate für die Zeichenkette `Hello` korrekterweise das Deutsche `Hallo` liefern, aber bereits die Präambel von TEX-Dokumenten zeigt, wie `\title{}`, `\author{}` und `\date{}` respektiv zu `\Titel{}`, `\Autorin{}` und `\Datum{}` übersetzt werden würden (Stand: 01.10.2025). Benanntes Tool zeigt sich zudem inkonsistent. Beispielsweise wird `\section{saw}` zu `\Abschnitt{Säge}` übersetzt und `\section{Introduction}` zu `\section{Einführung}` übersetzt.

Einzelne Lexeme menschlicher Sprache werden durch Freifläche (Whitespace) voneinander getrennt. Dies würde die Hoffnung aufwerfen, dass jegliche TEX-Syntax anhand von Sonderzeichen (bspw. \,{,},. . .) erkennbar wäre. Dies ist allerdings nicht der Fall, da sich z.B. Farben via `\definecolor{super light red}{rgb}{1,.5,.5}` definieren lassen könnten. Diese Zeichenkette hält das freistehende englisches Wort "light", bei welchem es fraglich ist, inwiefern es zu übersetzen ist. Hierbei entstehen zwei Optionen, welche beide die TEX-Syntax einhalten. Einerseits kann an jeder Stelle im Dokument das freistehende Wort zu "Licht" übersetzt werden oder an keiner. Dies wäre daran abzuwägen, ob das Wort in Kombination mit dem zuvorigem "super" und nachfolgendem "red" auftritt. Es könnte per se auch der Fall eintreten, dass die Wortfolge im menschlichen

Sprachegebrauch innerhalb des Dokumentes erfolgt. ("His jacket wasn't just red. It was this kind of *super light* red, which you'd typically only see when looking raw mutton"). Nach diesen syntaktisch richtigen Optionen, bleiben nur noch Syntaxbrechende. Sollte also an einer Stelle, an welcher die definierte Farbe `super light red` gemeint ist, das Wort "light" übersetzt werden, so würde die Farbe auf ihren initialen Standardwert von 0 für alle Farbkanäle (also: schwarz) gesetzt sein. Sollte ein Dokument einen schwarzen Hintergrund besitzen, wären diese textlichen Inhalte nicht mehr lesbar.

Bereits in den einleitenden Worten wurde klar, dass das Übersetzen von bestimmten Zeichenketten zu schwerwiegenden Brüchen in der TEX-Syntax führen könnte. Vor allem für die Einbindung anderer Dateien spielt dies eine herausragende Rolle, da jegliche Datei (via `include` oder `input` hinzugefügt) maßgebend für das entstehende Dokument werden könnte.

2.2 Komplexere Probleme

Neben den zuvor geschilderten sehr einfachen Problemen, welche sich auch unabhängig von TEX (und LATEX) zeigen könnten (denn z.B. ein Übersetzen von Hashtags im Social Media sollte keine neue Idee sein) und gelöst sein müssen (da ansonsten sehr einfache und rudimentäre Werkzeuge für eine Dokumentenerstellung verloren gehen, da man sich ohne diese simplem Formatierungsoptionen wieder auf einfache Textdateien berufen könnte).

Makros sind eine Möglichkeit mehrere TEX-Befehle zusammenzufassen. Vor allem in LATEX sind eine Vielzahl dieser bereits vordefiniert, jedoch handelt es sich bei diesen meist um Wörter der englischen Sprache ("meist": manche dieser englischen Wörter treten auch in anderen Sprachen auf, bspw. *paragraph*↔"Paragraph"). Sollte es einem TEX-User leichter fallen in der z.B. französischen Sprache zu arbeiten, so könnte dieser beispielsweise neue, französische Makros mit

```
\newcommand{\anglais}{This is some \textit{formatted} \texttt{english} \TeX{}-t}
```

erzeugen. Das vorige Beispiel zeigt zudem auf, wie Texte innerhalb von TEX-Makros "verschwinden" können und wirft die Frage auf, wann und wie solche Texte übersetzt werden sollten. Am sinnvollsten erscheint zunächst nur Zeichenketten zu übersetzen, welche sich mit der prominentesten Sprache des gesamten Dokumentes decken, welche allerdings nicht ohne weiteres bekannt ist. Selbst wenn in dem gesamten Dokument größtenteils englische Wörter vorliegen, ist eigentlich nur interessant, in welcher Sprache die reinen Strings (welche auf der PDF lesbar erscheinen) geschrieben sind. Selbst diese Information alleine ist theoretisch gesehen noch keine Grundlage für eine Aussage darüber, welche Sprache in solche einem Fall übersetzt werden müsste, da man hier Kenntnis des eigentlichen, entgültigen Dokumentes bräuchte, denn es könnte auch von Interesse sein, innerhalb eines größtenteils z.B. deutschsprachigen Dokumentes nur vereinzelte, englische Sätze zu übersetzen. Hierauf wird in Abschnitt 2.4 näher eingegangen, da sich dieses Problem zunächst recht einfach durch eine Auswahlmöglichkeit der Ausgangssprache (= die zu Übersetzende) lösen ließe.
Gleiches ist zu berücksichtigen, sollte das Kommando `\renewcommand` verwendet werden, wobei dieses allerdings noch ein wenig mehr zulässt. Hiermit ist man auch dazu in der Lage existierende Befehle der LATEX-Syntax zu ändern, wodurch ein `\Abschnitt{Einleitung}` ebenfalls valide LATEX-Syntax werden könnte, welche ein TEX-Compiler als `\section{Einleitung}` richtig interpretieren könnte, aber ein übersetzendes Programm könnte dieses womoglich in `\section{example}` überführen. Dies scheint zunächst kein Problem zu sein, jedoch hätte zwischen einem `\renewcommand{\section}{\Abschnitt}` genauso ein `\newcommand{\section}{\frac{1+\` stattfinden können, wodurch `\section{example}` nicht in einem Abschnitt mit Titel "example", sondern in $\frac{1+\sqrt{5}}{2}$example resultieren würde.

Umgebungen sind, wie der Name es vermuten lässt, einzelne Bereiche im Dokument, welche gesondert behandelt werden und für welche sich jegliche Einstellungen, wie z.B. Textfarbe, Textgröße, Schriftart, Font und vieles Weitere nur für eine solche Umgebung anpassen lassen. Einerseits kann man über geschwungene Klammern {} eine Umgebung einmalig betreten oder verlassen, möchte jedoch auch die Möglichkeit erhalten diese erneut zu verwenden und ihr verschiedene Parameter zu übergeben. Eine Definition einer Umgebung in der Präambel lässt dies zu, wodurch sich neben den in **??** aufgezeigten Problemen nicht nur für etwaige

Farboptionen und -einstellungen Strings aufzeigen, welche nicht übersetzt werden dürfen, sondern auch eigens (vom TEX-User) Ausgedachte, wie das an (Overleaf 2025) angelehnte Beispiel:

```
\newenvironment{boxed}[2][this is an example]
{
    \begin{center}
    Argument 1 (\#1)=#1\\[1ex]
    \begin{tabular}{|p|}
    \hline\\
    Argument 2 (\#2)=#2\\[2ex]
}
{
    \\\\\hline
    \end{tabular}
    \end{center}
}
```

Diese Umgebungen selbst sind zunächst nur von Interesse, wenn sie *default* Werte beinhalten, wie obiges `this is an example`, bei welchem es wünschenswert wäre, wenn ein Programm, welches TEX übersetzt, diese erfässt. Auffällig wird an dieser Stelle bereits, dass noch sehr viel mehr mit Umgebungen möglich ist, worauf in 2.3 näher eingegangen werden soll.

Pakete bieten eine TEX-Schnittstelle für die gesamte Welt! Zumindest rein theoretisch natürlich. Technisch gesehen bieten *packages* die Möglichkeit zuvor beschriebene Umgebungen und Makros in einer eigenen `.sty` zu bündeln, welche ihrerseits (vorrangig via) CTAN (jedoch auch auf jeglichem anderem Wege) zu anderen TEX-Usern übertragen werden könnte. Verschiedene Pakete könnten hierbei eine Vielzahl individueller Probleme aufwerfen, zunächst ist jedoch mehr ein Fokus auf solche zu setzen, welche die Arbeit anderer Programme involvieren. Sie in einem Dokument einzubinden ist recht leicht und funktioniert nur mit einer begrenzten Anzahl an Methoden (`requirepackage` und `usepackage`) und muss ihrerseits, genauso wie **??**, stets zur Kompilierzeit im System vorhanden sein.

TikZ ist zum Einen eine Möglichkeit in TEX zu malen, jedoch hauptsächlich dahingegen konszipiert in einem wissenschaftlichen Kontext verwendbare Diagramme mathematisch zu beschreiben oder auf Grundlage von Messwerten zu erzeugen. Die Syntax von Ti*k*Z und `pgfplots` kann innerhalb eines Dokumentes auch freistehende englische Wörter beinhalten, wie zum Beispiel in...

```
\begin{tikzpicture}[h!]
    \centering
    \begin{axis}[
        domain=-8:8
    ]
    \addplot{x};
```

```
    \end{axis}
\end{tikzpicture}
```

... bei welchem ein Übersetzen von "domain" Fehler produzieren würde, da Ti*k*, bzw. `pgf` von dem englischen Wort ausgeht, um die Grenzen des Plots bei -8 und $+8$ zu setzen. Jedoch besitzt Ti*k*Z noch einen größeren funktionalen Umfang. Wohingegen ein Erstellen und Nachbearbeiten von präzisen Graphiken in z.B. Adobe Photoshop, GIMP, Paint, Blender, ... zwar schnelle Korrekturen auf Pixelebene zulassen, ist jedoch kein "einfaches" Verschieben von z.B. einer Kante oder eines Knoten eines Graphen gegeben, geschweige denn ein Hinzufügen eines neuen Knoten zu einem Graphen. Dies ist in Ti*k*Z jedoch kinderleicht, da man sich hier nur um eine Beschreibung eines Graphen kümmern muss, welche sich leichter anpassen lässt, als ein(e) gesamte(s), existierende(s) Graphik oder Modell. Hierzu könnte man beispielsweise mit

```
\tikz \graph {
a -> b ->[green] {
        c,e,g
    };
    c ->[red] e,
    e ->[blue] g,
    a ->[yellow] g
};
```

den nichtsaussagenden Graphen in **??** erzeugen. Natürlich lassen sich auch zahlreiche andere Typen von Graphen erzeugen (Tantau 2013), wichtig ist jedoch nur wann und wie innerhalb dieser Texte auftreten könnten, welche interessant sein könnten, wenn man ein LaTeX-Dokument übersetzen möchte.

BibTeX wird genutzt um Zitationen/Referenzen/Literaturverweise innerhalb eines Einzelnen oder mehreren Dokumenten zu nutzen und zu verwalten. Die BibTeX-Notation selbst beläuft sich auf eine einfache JavaScript Object Notation `.json` und trägt mit nur einer Ausnahme nicht zu übersetzende Inhalte, wie den Autor, den Titel des Werkes (welcher in der Originalsprache stehen muss, andernfalls sollte die Übersetzung vom Autoren bestätigt werden), das Datum, einer URL, einer DOI und einer Angabe über die Art der Publikation, also ob das zitierte Werk aus einem Buch, einer laufenden Reihe/Serie an wissenschaftlichen Publikationen (bspw. *nature*, *science*, *ACM Computing Surveys*,...) oder einer Konferenz (oder Sonstigem) stammt. Neben diesen Angaben bleibt das Abstrakt eines zitierten Werkes interessant für einen Übersetzungsvorgang, sollte man davon ausgehen, dass im Anschluss entstehende, übersetzte `.tex` Dateien an einen neuen Autoren übergeben werden.

Mathematische Formeln selbst sind kein eigenes Paket, jedoch einer der praktischsten Use-Cases von TeX. Insbesondere für Menschen, welche sich eine handschriftliche Qualität und "Streichlust" (meint: das Durchstreichen auf dem Papier, sollte man sich verschrieben haben) mit der des Autoren (dieser Arbeit) teilen, sollte das digitale Medium TeX einiges an Aufwand ersparen und jegliche Herleitungen deutlicher und übersichtlicher machen. Hierzu gibt es wiederum mehrere denkbare Pakete, welche diesen bereits in

TEX inhärent verankerten "*math mode*" erweitern oder vereinfachen können. Dabei muss man sich jedoch zunächst wieder vor Augen führen, welche Inhalte man darstellen möchte und inwiefern ein Programm, welches `.tex` übersetzt bestimmte Inhalte übersetzen muss. Dadurch wird schnell klar, dass Pakete, wie zum Beispiel `amsmath` und Untergeordnete nicht sonderlich relevant werden, da sie ihrerseits nur Befehle beinhalten, welche der "normalen" TEX-Syntax obliegen. Lediglich Pakete wie zum Beispiel `theorem` sind hierbei (im Kontext der gegebenen Aufgabenstellung) von näherem Interesse, da sie sich dadurch auszeichen, dass sie bereits zur (Kompilierzeit in der) Präambel auf eigener Syntax basierend rein textliche Strings definieren. Aus jeglichem Mathematikmodul, -seminar, -kurs, -unterricht und dergleichen sollten einige geläufige Teile von Herleitungen oder Beweisen bekannt sein sowie übliche Terminologie für solche Sektionen. Beispielsweise müsste das Wort "Definition" einem jeden Studierenden einen kalten Schauer den Rücken herunterlaufen lassen, muss es glücklicherweise in diesem Moment allerdings nicht, da in TEX (bzw. `thesis`) dieses Wort nur ein Mal (und auch vor dem eigentlichen Dokument) definiert wird und zwar in der Form `\newtheorem{definition}{Definition}`. TEXnisch gesehen passiert hier nichts weiter, als eine Definition einer neuen Umgebung, welche sich danach (innerhalb des Dokumentes) darin äußert, dass das Wort in den Zweiten geschwungenen Klammern breit gedruckt wird und als Wiedererkennungsmerkmal für (hier:) eine Definition dienen soll. Normalerweise sollte es in der Mathematik nur wenigen Definitionen bedürfen. Anders ist dies jedoch bei Beweisen, von welchen überabzählbar viele gebraucht werden und gebraucht werden könnten. Eine sich hierbei aufzeigende Schwierigkeit könnte es werden einen Überblick über solche zu schaffen (innerhalb einer z.B. Vorlesung) ohne diesen einen Weg mitzugeben schnellstmöglich wiederauffindbar zu werden, ohne den gesamten Beweis zu denotieren. Hierzu kann man die Fähigkeit von `theorem` nutzen, dass man z.B. Theoreme/Behauptungen einem TEX-Abschnitt zuordnen kann und auch andere `theorem`-Umgebungen Anderen desselben Paketes (über eckige Klammern nach (um ein "`.zahl`" zu erzeugen) oder vor der Benennung dieser namensgebenden Definition (um ein "*nächsthöhere `theorem`-zahl*" zu erzeugen)).Weitergehend sind die eigentlichen Inhalte von mathematischen Umgebungen für eine Übersetzung zwischen Menschensprachen nicht relevant, da es sich bei jeglichen Strings innerhalb dieser Umgebungen nur um wirkliche Wörter oder LATEX-Befehle handelt, welche normalerweise anhand mathematischer Symbole oder dem Backslash zu erkennen sind (Lamport 2003). (Bemerkung zur letzteren Zitation: In dem Werk selbst geht es zwar nicht um TEX selbst, jedoch nutzen die angegebenen Seiten die LATEX-Makros, schließlich stammt das Werk von dem `La` aus LATEX).

Quelltexte lassen sich mit Hilfe der Pakete `minted` und `lstlisting` in einem TEX-Dokument darstellen und formatieren. Dies scheint auf den ersten Moment noch kein allzu großes Problem zu sein, da man diese anhand der jeweiligen Umgebungen `\begin{minted}` und `\begin{listing}` erkennen könnte und auch aus z.B. den Einstellungsmöglichkeiten für die Sprache, welche formatiert werden soll nicht zu übersetzende Token kennen könnte. Hierbei ist nunmehr interessant, ob in diesen Quelltexten nicht eventuell Zeichenketten verankert sind, welche für eine Übersetzung interessant wären. (Wird beispielsweise ein String ausgegeben, welcher von `Hello World` zu `Hallo Welt` übersetzt werden könnte/sollte?) Darüber hinaus wäre es in Erwägung zu ziehen TEX-Quellcode selbst in einer solchen Umgebung darzustellen, oder aber HTML, bei welchem fraglich ist, bis zu welchem Grad die richtigen Zeichenketten erfasst werden und nicht versehentlich ein `<div stlye=\color:red">` zu `<div stlye=\Farbe:rot">` übersetzt werden würde. (Hier gerät man an einen ähnlichen Punkt, wie 2.3 an, nur in einem recht spezifischen Kontext.) Eine unabhängige Google-Suche

ergab interessanterweise **??**

Eigene Pakete innerhalb LaTeX zu erstellen ist natürlich auch möglich, wobei danach zu differenzieren ist, ob man ein *Klasse* oder ein *Paket* schreiben möchte. Der Unterschied wird danach festgelegt, ob sich die Befehle innerhalb der jeweiligen `.cls` oder `.sty` mit jeder Art (Klasse, das Dokument aus logischer Sichtweise hinsichtlich Aufbau/Struktur) verwenden lassen. Ist dies der Fall spricht man von einem Paket `.sty`, andernfalls von einer Dokumentenklasse `.cls` (The LaTeX-Project Team 2025). Es handelt sich ihrerseits jedoch auch wieder um reine Textdateien, in welchen man zuvorig geschilderte komplexe (und folgende speziellere) Probleme verschwinden lassen könnte. An sich handelt es sich bei diesem Problem also um eine sehr spezifische Erweiterung des letzten simplen/einfachen Problemes in **??**.

2.3 Spezielle Probleme

Höhere Vernestungsgrade können in TEX auf verschiedenste Arten und Weisen entstehen, daher zunächst eine kurze Erinnerung, was der Begriff in der Informatik und Mathematik meint. Recht einfach wird die Problemstellung der Vernestung an einem Beispiel verdeutlichbar: Ist $(((((((1+1)-1)-2)*3)/4)\%5)/6)*7)$ leicht auszuwerten? Eine hohe Vernestung bedeutet mehrerlei Klammern innerhalb von (einer) Klammer(n) vorzufinden. Inwiefern kann dies in der Informatik auftreten? Betrachtet man bestimmte syntaktische Elemente als klammerähnlich (wie z.B. ein `for (condition) {befehl ()}` in C, wobei es sich um tatsächliche Klammern hält), zeigt sich schnell eine Unlesbarkeit in Quelltexten auf, sollte man den Grad dieser Verklammerung erhöhen. Wäre z.B.

```
if(x)
{
    if(y)
    {
        if(z)
        {
            if(k)
            {
                if(l)
                {
                    if(m)
                    {
                        if(a)
                        {
                            if(b)
                            {
                                if(c) return true;
                            }
                        }
                    }
                }
            }
        }
    }
}
```

ein leicht lesbares, verständliches und demnach wartbares C-Code-Fragment? (Nein, könnte jedoch syntaktisch richtig sein). Dies ergibt mathematisch/logisch die Aussage: $return = 1 \wedge c \wedge b \wedge a \wedge m \wedge l \wedge k \wedge z \wedge y \wedge x$. Auch wenn solche hohen Vernestungsgrade nicht erstrebenswert sind, sollte die wohlmögliche Existenz dieser nicht missachtet sein. Dies wirft die Frage auf, wo solche Vernestungen schnellmals auch unbemerkt entstehen könnten.

In Tabellen ist eine Vernestung zunächst nicht auszuschließen. So kann es schnellmalig der Fall sein, dass man innerhalb einer Zelle einen numerischen Wert mathematisch abbilden möchte, jedoch eine physische Einheit textlich formatiert sehen will (ohne sich hierbei dem `siunitx` Paket zu bedienen). Nun sollte man zumindest davon ausgehen, dass sich Tabellen, wie man sie in wissenschaftlichen Veröffentlichungen vorfinden kann, zunächst in einer Form wie beispielsweise:

```
\begin{table}[h!tb]
    \centering
    \begin{tabular}[l r]
        \toprule
            distance $[$m$]$ & time $[$s$]$
        \midrule
            $400$ & $60$   \\ % starting at a fast pace
            $800$ & $121$  \\
            $1200$ & $183$ \\
            $1600$ & $242$ \\
            $2000$ & $300$ \\ % starts to sprint
            $2400$ & $350$ \\
            $2800$ & $420$ \\ % starts feeling fatigued
            $3200$ & $470$ \\
            $3600$ & $550$ \\ % fatigue ultimately looses time from this point on
            $4000$ & $710$ \\
        \bottomrule
    \end{tabular}
    \caption{Track-record of a fictional runner's pace on \today. This table
    ↪  requires the packages \texttt{caption} and \texttt{booktabs}!}
    \label{tab:1}
\end{table}
```

. . . vorliegen würde, wobei zu übersetzende Wörter frei stehen. Jedoch wird hier eine Hürde der sprachlichen Übersetzung insgesamt erkenntlich, auf welche in 2.4 kurz eingegangen wird. Diese recht einfache Nutzung ist recht handhabbar, kratzt allerdings nur an der Oberfläche der Möglichkeiten, welche TEX bieten kann. So ist es auch denkbar, dass man eine Tabelle erstellen möchte, in welcher eine Spalte einen erwarteten Funktionsverlauf einer Größe gegenüber z.B. der Zeit darstellt, eine zweite Spalte real bemessene Werte zu verschiedenen Zeitpunkten, eine Dritte in welcher die Werte über den erwarteten Verlauf gelegt werden und eine Letzte, in welcher nun Mittelwerte, Varianz und weitere Bemerkungen festgehalten werden. (Sollte zuvorige Beschreibung etwas irritierend sein, dann dient refapp:functiontable als Veranschaulichung).

Zwischen verschiedenen Umgebungen kann (quasi-) beliebig hin- und hergewechselt werden. Dies ist zunächst keine neue und wichtige Information, spielt aber insbesondere auf die Darstellung von mathematischen Inhalten ein. So ist es nicht unbedingt erforderlich, dass nach dem Betreten einer mathematischen Umgebung (via z.B. `$`, `$$`, `\(`, `\[`, sowie `\begin{equation*}`, `\begin{align*}`, `\begin{gather*}` mit und ohne `*`, . . .)

diese zwangsweise direkt wieder verlassen werden muss, bevor man wieder textliche Inhalte produzieren und beliebig formatieren kann. Die gewöhnlichen Kommandos zum Erzeugen von breit gedruckten, kursiven oder anderweitigen Texten (**textbf**, *textit*, textrm, TEXTSC, textsf, texttt, ...) reichen hierzu zunächst... lassen es ihrerseits jedoch zu, dass man wieder in eine mathematische Umgebung wechselt. Ein valides TEX-Beispiel:

```
\begin{align*}
    4x &= 2 \\
    x &= \frac{1}{2} \textrm{\@\ldots zeigt zugleich, dass $\frac{1}{2}\times 4=2$
    ↪  ist}
\end{align*}
```

(produziert:)

$$\begin{aligned} 4x &= 2 \\ x &= \frac{1}{2} \ldots \text{zeigt zugleich, dass } \frac{1}{2} \times 4 = 2 \text{ ist} \end{aligned}$$

Dies alleine ist natürlich noch keine sonderlich hohe Vernestung, zeigt jedoch die Vorgehensweise auf, mit welcher solche Vernestungen erzeugt werden können. Nun könnte argumentiert werden, dass in der herkömmlichen Dokumentenklasse `article` solche *in-line* Wechsel nur bedingt oft vorkommen und behandelt werden müssten, da ansonsten zu lange Zeilen nicht mehr innerhalb eines Dokumentes angezeigt werden würden. Hierfür existiert allerdings ein Workaround und zwar die Dokumentenklasse `standalone`, welche theoretisch gesehen unendlich lange Dokumente erzeugen kann, selbst wenn diese nur eine sehr lange Zeile sind.Solange auf jeden Wechsel in eine mathematische Umgebung ein Wechsel (an geeigneter Stelle) aus dieser heraus folgt und das Gleiche auch für Text-Umgebungen vergewissert ist, gibt es an sich keinen Grund ein Limit bei dieser Art von Vernestung zu setzen.

Kommentare sind ein aus jeglicher Programmiersprache bekanntes Feature um die Funktionsweise eines Quellcodes zu erklären, damit das Nachvollziehen dieses Codes einem anderen Entwickler erleichtert wird. Zu erwarten wäre zunächst nur eine reine Nutzung von Kommentaren für ihren ursprünglichen Zweck, also in der Form:

```
This is some text.\\[7pt]
\noindent This is some more text.\\
% This is a comment
\noindent Comments won't be printed!
```

Andererseits ist es auch nicht unüblich Code auszukommentieren, insofern dieser nicht funktioniert, da er syntaktische Fehler (ggb. der jeweiligen Programmiersprache) beherbergt.

```
%\begin[environment]
%    Testing a new environment!
%\end[environment]
% Why doesn't this work???
```

Fraglich wird hierbei, inwiefern auch hier mitunter komplexere Beispiele erfasst werden. Denkbar ist nämlich neben den einfacheren Beispielen, dass sich Ti*k*Z Graphiken, Quellcode anderer Sprachen, Kapiteltitel (welche übersetzt werden sollten) oder gar Kommentare in Einzelnen dieser TEX-Quellcodes befinden.

Definitionen sind die technische TEX Grundlage von LATEX Makros. Neben der Möglichkeit einzelne Zahlen (und Zeichenketten) mit Hilfe von ihnen zu definieren, kann man sie ebenfalls dafür nutzen eigene Logik in einem Dokument zu integrieren.

```
\def\a{Nummer}
\def\b{Zahl}

\ifx\a\b
This sentence will be expected to be read, if this document has been translated
↪  into english.
\else
Dieser Satz wird zu sehen sein, wenn dies ein deutsches Dokument ist.
\fi

\a\b\a\b
```

Wenn nun die beiden Definitionen übersetzt werden würden, dann würden die inhaltlichen Aussagen der beiden Sätze stimmen. Dreht man jedoch die Logik der beiden Sätze um (*has*→*hasn't* und *ein*→*kein*), dann würde eine Übersetzung die inhaltlichen Aussagen der Sätze verfälschen. Ob in solchen Fällen `def`'s übersetzt werden sollen, wäre unvorhersagbar, sollte man keine Informationen darüber erhalten können, inwiefern sie logisch genutzt werden. Diese Information liegt jedoch im Dokument vor, zumindest rein theoretisch gesehen und es wäre denkbar, dass ein übersetzendes Programm erst prüfen könnte, ob sowohl `\def` genutzt wurde und danach diese definierten Makros mit einer Logik verknüpft wurden, damit danach innerhalb dieser Verknüpfung (bspw. die obige `if-else`) danach geforscht werden kann, inwiefern dies Rückschlüße darauf liefern kann, ob übersetzt werden soll oder nicht.

Catcode und Unicode hätten eigentlich nicht nur einen eigenen Paragraphen, sondern wahrscheinlich ein ganzes Kapitel verdient. Jedoch lässt sich die Funktionsweise von `catcode` sehr schnell auf den Punkt bringen. Jedem Unicode-konformen Zeichen (welches in einer Textdatei auf einem Computer, bzw. innerhalb der TEX-Engine landen kann) könnte eine Bedeutung für den TEX-Parser zugewiesen werden. Die Buchstaben `c` und `b` könnten von ihrer Bedeutung mit den Zeichen `{` und `}` gleichgesetzt werden.

```
{
    \catcode99=1 % c={
    \catcode98=2 % b=}

    c\Large This is large textb\\
}
and this is regular one.
```

Geht man bedacht an die Sache heran und definiert passend jeweilige Makros um:

```
{
    \catcode99=1 % c={
    \catcode98=2 % b=}

    c\Large This is large textb\\
}
and this is regular one.
```

So lassen sich einzelnen Zeichen völlig neue Bedeutungen für die TEX-Engine geben! Vermutlich könnte man sogar dazu in der Lage sein, dass man jegliche Zeichenkette zu einem denkbaren und beliebig interpretierbaren Makro macht, sodass selbst:

```
afcq2h9d.bcshd<
```

äquivalent zu

```
\section{begin}
```

werden könnte. Hierbei stellt sich jedoch die Frage, inwiefern dies sinnvoll ist und eine Erleichterung in der Erstellung von Dokumenten bietet.

2.4 Weitere Schwierigkeiten

Beabsichtigt ist dieser Abschnitt nicht in der Reihe von Problemen aufgefasst, sondern als Schwierigkeit(en) formuliert, da man sich hier von den Problemen abwenden würde, welche in der TEX-Syntax auftreten und bei sprachliche Hürden angelangt, welche sich für und zwischen verschiedenen Sprachen zeigen könnten.

Mehrdeutigkeiten innerhalb einer Sprache führen unter Umständen zu missverständlichen Übersetzungen. Ein recht einfaches Beispiel bietet bereits das sehr allgegenwärtige Wort "ungerade", welches je nach Kontext als "schief" interpretiert werden könnte, oder aber für die Aussage, dass eine Zahl modulo 2 nicht 0 ergibt. Weiterhin existieren selbst sprachunabängig Mehrdeutigkeiten für bestimmte Wörter/Konstante. So muss zum Beispiel für eine "Meile" je nach Kontext abgewogen werden, ob es sich um eine Seemeile oder eine Landmeile handelt (zwischen welchen immerhin rund 200 Meter Unterschied bestehen).

Redewendungen sind eine Art und Weise anderweits nicht beschreibbare Inhalte und Situationen zu schildern. Jedoch unterscheiden sich diese je nach Sprache, sodass das deutsche "Ich glaub ich spinne" im Englischen nur Verwirrung schüren würde, sollte es Wort für Wort übersetzt worden sein.

Abkürzungen

SI-Einheiten sind die eigentlichen Grundeinheiten, auf welche man versucht physikalische Größen (genauer: für diese hergeleiteten Einheiten) zurückzuführen. Mit sehr wenigen Ausnahmen sollte davon auszugehen sein, dass diese international Verwendung finden. Einzig und allein Distanz- und Masseangaben *könnten* je nach Sprache variieren, wie das imperiale Maß gegenüber dem metrischen Maß (e.g., Zoll und Meile ggb. Zentimeter und Kilometern, `lbs` ggb. `kg`; bei welchen es sich zwar nicht um die *eigentlichen* Grundeinheiten handelt, jedoch in dieser Form in der realen Größenordnung miteinander vergleichbarer werden).

Wirrer Sprachwechsel meint ein rapides Springen zwischen verschiedenen Menschensprachen innerhalb eines Dokumentes. Die Fragestellung hierbei ist, inwiefern ein sprachlicher Wechsel innerhalb eines Dokumentes erfasst wird, sollte eine automatische Spracherkennung der Ausgangssprache stattfinden. Dabei können verschiedenste (theoretisch: überabzählbar viele) Fälle auftreten, unter welchen z.B. Wechsel aus dem Deutschen in das Englische an beliebieger Stelle im Dokument, satzweisige Wechsel zwischen zwei und mehreren Sprachen, sowie ein nur kurzfristiger Wechsel in eine Sprache, innerhalb eines ansonstig monolingualen Dokumentes, welche allerdings Lexeme dieser beinhält (bspw.: ein norwegisches Dokument beinhält ein dänisches Zitat).

Whitespace lässt sich in TEX nicht nur mit ´ erzeugen. Die Zeichen, bzw. Zeichenketten `\ `, `\@` und `~` können genauso Freifläche zwischen einzelnen Strings produzieren.

3 Technologischer Stand und denkbare Lösungswege

3.1 Generische, fundamentale Technologien

Google Translate wurde während der Erläuterung im vorherigen Kapitel herangezogen, da sich mit dieser Ressource recht einfach typische Fehler bei unbedachter Übersetzung von LaTeX Dokumenten aufzeigen lassen. Solche Software erwartet als einen Input immer ein Lexem (Wort) einer menschlichen Sprache, welche (aktuell) keine "Sonderzeichen", bzw. Symbole beinhalten (gemeint sind z.B. `#,\,§,=,+,-,...`). Ein Mensch kann solche+ Zeichen im Lesefluss ignorieren, ein Programm (bzw. ein Computer) kann dies ohne (ein) Weiteres jedoch nicht.

Jedoch basieren heutige Technologien zur Sprachübersetzung nicht mehr auf Programmen, welche einen festen Input mit syntaktischen Vorgaben erwarten, sondern auf weitaus mächtigerer Software, welche je nach erforderlichem Use-Case eine andere Art an Input erwartet. Als Endnutzer würde man sich zunächst jegliche Optionen offen halten wollen, wodurch man recht schnell bei den Technologien größerer und bekannter Anbieter angelangt, welche Anwendungs- und Nutzerschnittstellen für ihre Sprachmodelle anbieten. Von den Dingen, welche aus marktwissenschaftlicher Argumentation heraus direkt weniger vielversprechend sein *müssten* (da diese Anbieter andere Software zur Dokumentenerstellung produzieren, welche sie verständlicherweise gegenüber einen kostenfreien Technologie wie TeX in den Vordergrund stellen möchten), wird zunächst noch nicht abgesehen, sondern jegliche denkbare Technologie hinsichtlich ihrer potentiellen Möglichkeiten betrachtet.

ChatGPT Der von OpenAI präsentierte statistische Ansatz zur Entwicklung einer künstlichen Intelligenz ist ein im ersten Moment naheliegender Ansatz, da man sich hier erhoffen könnte, dass diese Technologie mit Hilfe eines passenden "Prompting" auf längere Zeit gesehen sowohl qualitativ hochwertige Übersetzungen erzielt, als auch geschickt jede TeX-Syntax geschuldete Hürde umgeht. Allerdings scheitert dieser Ansatzbereits konzeptionell, denn der TeX-Compiler (jeder) selbst wird dazu in der Lage bleiben *nur* reine Zeichenketten von Befehlen, Makros und Ähnlichem zu unterscheiden, da diese ansonsten nicht wie gewünscht in einem Dokument als solche Zeichenkette vorzufinden wären. Da dieses Programm selbst auch deterministisch arbeitet (und arbeiten muss), benötigt man an dieser Stelle noch keine Einbindung einer potentiellen Fehleranfälligkeit.

DeepL

Microsoft Translate

Google Cloud Translate

3.2 Denkbare Workflows

Setzt man sich vor die zunächst etwas einfacherere und bisher noch offene Problemstellung, wie man überhaupt alle textlichen Inhalte eines LaTeX Dokumentes `computationally` erfasst,sind verschiedene Wege denkbar. Ausgehend von einem TeX "Main"-Code könnte man diesen (a) übersetzen und dem zusehen, dass die TeX-Syntax erhalten bleibt(b1) einen TeX-Compiler dahingegen verändern, dass dieser zu übersetzende Strings

zusätzlich in eine eigene Datei ausgibt, auf deren Grundlage übersetzt werden kann (b2) den TEX-Kompilier-Prozess so zu ändern, dass während diesem jegliche textliche String-Token an ein übersetzendes Tool gesendet werden oder (c) in ein anderes Format zur Beschreibung von Dokumenten überführen, wie bspw. (HTML, XML, PDF, . . .)und ausgehend von diesem übersetzen.

Workflows, von welchen abzusehen ist In erster Linie wäre es durchaus denkbar, dass man den Fakt ausnutzt, dass ein TEX-Compiler jegliche rein/wirklich textlichen Strings (welche in kompilierter PDF als diese angezeigt werden) selbstständig und während der Kompilierung an eine API eine der zuvor gelisteten Übersetzungs-Tools sendet und danach wieder im Dokument integriert. Dies mag für Workflows funktionieren, in welchen TEX-Dokumente nur selten kompiliert werden, jedoch bieten heutige Mittel die Möglichkeit eines Live-Rendering, bei welchem sehr schnell und öfters kompiliert wird (da ein Endnutzer schnellstmöglich dessen Änderungen im Dokument sehen möchte). Dies impliziert, dass bei jeder Kompilierung der TEX-Datei auf eine entsprechende und kostenpflichtige API zugegriffen werden würde, wodurch schnellig und nicht direkt bemerkte Kosten anfallen könnten, wenn man nicht bedenkt, dass man auch nur die wirklich veränderten Teile (seit letzter Übersetzung) an ein Übersetzungstool senden müsste. Hierbei wird jedoch fraglich, inwiefern der gesamte Kontext des Dokumentes erhalten bleibt, bzw. wie man diesen live bestimmt. Was geschieht, wenn größere Textblöcke gelöscht werden und dadurch Kontext für spätere Textblöcke verloren geht? (Im Sinne: Wir sprechen von Zeile 1 bis x von Cybersecurity und löschen Zeilen 1 bis y, wobei y¡x. Soll und muss dann der Kontext Cybersecurity erhalten bleiben? Woher kann und soll ein live-agierendes Programm dies wissen, wenn es immer nur Zugriff auf die in diesem Moment vorliegenden Texte hat, in welchen *nun* kein Cybersecurity-Kontext mehr vorliegt. . . und demnach nicht mehr in diesem Kontext richtige Übersetzungen gewählt werden können.)Wann und wie kann man also, rein von der Menge des editierten/gelöschten Textes absehen, wann und wie sich der Kontext eines Dokumentes ändert? Die Logik innerhalb eines Dokumentes könnte sich bereit dass vollständig ändern, sollte man nur ein (sehr frühes) "nicht" löschen, wodurch die Logik fortan inversiert wäre.

Nach TeX entstehende PDF könnten theortisch gesehen auch als Grundlage innerhalb eines automatischen Workflows denkbar sein. Denkbar sind Technologien, welche z.B. eine PDF in HTML oder XML zerlegen und diese als Grundlage nutzen, um eine automatische Übersetzung, für welche bekannte Browser eine Unterstützung bieten, zu nutzen. Diese Übersetzten HTML danach wieder in PDF zu übertragen ist noch leichter, da dies die herkömmlichen Browser ohnehin können. Und wem das Kompilieren auf diesem Weg zu lange dauert, darf sich in der Zwischenzeit dem Dokument in HTML/XML bedienen, welches ein Browser anzeigen kann. Sollten hiernach noch kleinere Änderungen von Nöten sein, so müsste nur noch eine entstandene, übersetzte PDF nur noch in den erwähnten Überlappungen innerhalb Ti*k*Z aufgrunde von unabdingbaren sprachlichen Verhältnissen angepasst werden, was dann einmalig in HTML erfolgen müsste. Hierbei könnten dann jedoch auch menschliche Spracheditoren den entstandenen Dokumenten einer Kontroll-/ Korrekturlesung unterziehen, sollte sich solche Menschen finden lassen. Dies erscheint in erster Betrachtung nachvollziehbar und logisch, sieht man jedoch genauer hin, so werden Schwierigkeiten bei dieser Herangehensweise offensichtlich, da man hier wohl kaum von einer gänzlich automatischen Übersetzung von LATEX-Dokumenten sprechen kann. Aus abstraktem Blickwinkel sind hier zwei Workflows denkbar. Zum einen könnte man TEX und Ti*k*Z geson-

dert voneinander betrachten, indem man sämtliche Ti*k*Z-Graphiken einzeln in idealerweise skalierbare Vektorgraphiken kompiliert und innerhalb einer nach HTML exportierten TEX-Datei als solche einbindet. Sämtliche textliche Strings sollten dann in dieser SVG vorzufinden sein, genauso wie es der Fall in XML ist (wodurch man sich hier auf bestimmte, und anhand eckiger Klammern identifizierbaren Tags, berufen kann, siehe: `https://svgwg.org/svg2-draft/text.html#TextElement`).

3.3 Existierende Ansätze

Paper Ohri und Schmah

PlasTeX und dann DeepL

TransLaTex

MathTranslate

Die TEX-Compiler Familie und rein innerhalb TEX-basierte Ansätze sind durchaus denkbar, erfordern jedoch einen höheren menschlichen Aufwand. Auf diese Art und Weise wäre eine Trennung von TEX und menschensprachlichen Texten direkt und ohne weiteres Nachdenken gegeben, sodass hierbei zwar Verluste hinsichtlich der Kompilationszeiten entstehen könnten, jedoch die gegebene Aufgabe technisch gesehen erfüllen könnten. Bemerkt sei hierbei, dass solch ein Ansatz natürlich bei Live-Editoren, wie beispielsweise Overleaf weniger geeignet wären, da diese für ihr Live-Rendering von PDFs bei jeder Veränderung des TEX-Dokumentes

Das translate package und dessen Glossare anpassen

Einen Compiler anpassen wäre möglich, sodass dieser alle Textstrings (hierbei: abhängig vom Token-Typ, welchen der TEX-Parser liefert) in z.B. YAML überführt, welche ihrerseits als Input für eine KI genutzt werden könnte, sodass mit Recht wenig Kontext weiß, welche Token sie übersetzen soll und in welchem Kontext sie stehen. Jedoch braucht dies nicht unbedingt tiefersitzend in einem spezifischen Compiler verankert integriert werden, sondern ließe sich auch, wie vorangegangene Ansätze des Abschnittes es zeigten, auch gesondert lösen. Daher sollte zunächst die Frage geklärt werden, inwiefern es sinnvoll ist große, komplexe und dadurch nur mit hohem Zeitaufwand nachzuvollziehnde Technologien als nicht-offizieller Mitentwickler zu verändern, sodass bei jedem offiziellen, neuen Release dieser Software inoffizielle Änderungen "mitgeschliffen" werden müssten, oder ob es nicht vorzuziehen wäre, dass man ein gesondertes Problem getrennt löst, sollte dies, wie hier, möglich sein.

4 Eigenständigkeitserklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbstständig angefertigt und ohne fremde Hilfe verfasst habe. Dazu habe ich keine außer den von mir angegebenen Hilfsmitteln und Quellen verwendet und die den benutzten Werken inhaltlich und wörtlich entnommenen Stellen habe ich als solche kenntlich gemacht. Ich versichere, dass die eingereichte elektronische Fassung mit den gedruckten Exemplaren übereinstimmt.

Rostock, den 02.12.2025

Hendrik Theede

Literaturverzeichnis

Knuth, D. E. (1986), *The TeXbook, ISBN: 9780201134476*, Addison-Wesley Professional.

Lamport, L. (2003), *Specifying Systems, ISBN: 9780321143068*, Addison-Wesley. https://lamport.azurewebsites.net/tla/book-21-07-04.pdf oder https://lamport.azurewebsites.net/tla/book.html (last Access: 04.10.2025).

Overleaf (2025), 'Environments - overleaf, online latex editor', https://www.overleaf.com/learn/latex/Environments (last Access: 03.10.2025).

Tantau, T. (2013), 'Tikz and pgf: Manual for version 3.1.11a', https://pgf-tikz.github.io/pgf/pgfmanual.pdf (last Access: 03.10.2025).

The LaTeX-Project Team (2025), 'Latex for package and class authors', https://www.latex-project.org/help/documentation/clsguide.pdf (last Access: 03.10.2025).

Unicode Consortium (2025), 'Unicode: The world standard for text and emoji', https://home.unicode.org/ (last Access: 06.10.2025).

A Anhänge

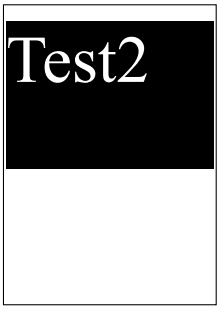
A.1 Fontskalierung auf Webseiten

Beispielsweise produziert die folgende HTML-Notation bei einer Skalierung im Browser von 120 Prozent (Abbildung 2a) und 50 Prozent (Abbildung 2b) jeweilig zwei verschiedene PDF (unter welchen nur Zweitere alle textlichen Inhalte offenbart). Ähnliches kann auch innerhalb TEX geschehen, sollte

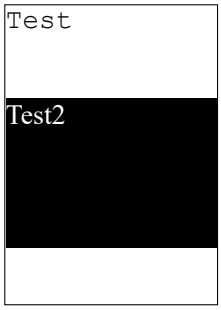
```
<html>
    <head>
        <title>Example</title>
        <style>
            /*formatting options are: none and black*/
            .t{
                font-size:13em;
                height:50%;
            }
            /*formatting option: none = no background, black, courier*/
            .t#none{
                font-family: 'Courier New', Courier, monospace;
            }
            /*formatting option: black = black background, white, serif*/
            .t#black{
                background-color:black;
                color:white;
                margin-top: -2em;
            }
        </style>
    </head>
    <body>
        <div class="t" id="none">Test</div>
        <div class="t" id="black">Test2</div>
    </body>
</html>
```

Abbildung 1: HTML-Beschreibung einer Webseite mit zwei Textflächen

Abbildung 2: Um die Dokumente von der restlichen Papierfläche abzugrenzen wurden schwarze Rahmen mittels Ti*k*Z hinzugefügt.



(a) Zuvorige HTML-Beschreibung liefert bei einer Browser-Skalierung von 120% obige Graphik



(b) Zuvorige HTML-Beschreibung liefert bei einer Browser-Skalierung von 50% obige Graphik