

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

Abstrakt


placeholder

Inhaltsverzeichnis

1	Einleitung	1
1.1	Hintergrund	1
1.2	Thematische Abgrenzung	1
2	Problemfälle	3
2.1	Herangehensweise	3
2.2	Elemente in einem T _E X-Quellcode	6
2.2.1	Die Präambel	6
2.2.2	Kommandos	6
3	Stand der Technik	16
3.1	Anforderungen	16
3.2	Denkbare Ansätze	16
3.3	Existierende Ansätze	16
3.3.1	Testverfahren	16
3.3.2	Durchführung	16
3.3.3	Auswertung	16
3.4	Grenzen der Lösungen	16
3.5	Takeaways	16
4	Eigenständigkeitserklärung	17
	Literatur	18
A	Anhänge	19
A.1	Fontskalierung auf Webseiten	19



1 Einleitung

1.1 Hintergrund

 Die schnellstmögliche und einfache Erstellung von Dokumenten beliebiger Natur (formlos) wird heutzutage oftmals über Produkte bekannter Anbieter abgewickelt (bspw. Microsoft's Word, PowerPoint, ... und die vergleichbaren „LibreOffice“-Software, sowie die korrespondierenden Apple-Produkte). Unterliegen Dokumente allerdings strengeren stilistischen Vorgaben (bspw. bei wissenschaftlichen Veröffentlichungen) entsteht der Vorteil, dass sich diese Vorgaben wie ein Regelsatz behandeln lässt, aus welchem sich bestimmte, vorprogrammierte Dokumentenstrukturen definieren lassen. Hierzu existiert bereits ein geläufiges System welches den Namen \LaTeX trägt (mit dem *La* nach einem der ursprünglichen Entwickler Lamport (1994)), welches selbst auf dem von Knuth (1986) entwickelten Zeichensetzungs-System und der verbundenen Programmiersprache \TeX basiert.

Die \TeX -Syntax selbst basiert auf englischen Begriffen, allerdings ist nicht davon auszugehen, dass nur englischsprachige Menschen \LaTeX und \TeX nutzen werden. Quelltexte und Dokumentenbeschreibungen werden also nicht immer in einer rein englischsprachigen Form vorliegen (z.B. `\chapter{Erstes Kapitel: Einleitung}` oder der Quelltext dieses Werkes). Ein Zurückführen solcher Dokumente in die englische Sprache ist einfach, insofern ein Verständnis der deutschen Sprache besteht (`\chapter{First Chapter: Introduction}`). Die Rückrichtung zeigt sich allerdings genau dann problematisch, sollte ohne Vorkenntnisse von \TeX (bzw. dessen syntaktische Elemente) bestehen. In genanntem Beispiel würde dann das Wort `chapter` aufgegriffen werden und die Zeichenkette `\Kapitel{Erstes Kapitel: Einleitung}` entstehen (ohne weitere, mögliche Anpassungen entsteht hier keine Kapitelüberschrift mehr. Das erstere „Kapitel“ würde ignoriert werden und die innerhalb der Klammern stehende Zeichenkette als einfacher Fließtext gedruckt werden).

1.2 Thematische Abgrenzung

 Bereits Shannon (1948) beschäftigte sich mit theoretischen Grundlagen der Darstellung und Übertragung von Informationen, insbesondere der menschlicher Sprache und Kommunikation. Heutige maschinelle Systeme zu diesem Zweck (bspw. ChatGPT, DeepL, Gemini und co.) wirken zunächst wie „magische“ Blackboxen, arbeiten jedoch auf Grundlage von statistischen Modellen. Spricht man hier von Magie, dann ist jeder Mitarbeiter eines Wetterdienstes oder der Klimaforschung „bezaubernd“. Das zugrundeliegende Konzept kann jedoch sehr schnell auf den Punkt gebracht werden: Eine KI erhält einen Input, für welchen ein bestimmter Output erwartet wird (Beispiel: „Einfügen“ als nächstes Wort eines zu übersetzenden Satzes und „insertion“ im Kontext innerhalb des Satzes als Erwartung (substantiviertes Verb)), und produziert einen Output, welcher mit dem Erwarteten abgeglichen wird. Sollte das Resultat von der Erwartung abweichen (z.B. „insert“ entstehen), so kann dieser Fehler erkannt werden und im Modell dazu beitragen, dass (gegeben einer bestimmten, sequentiellen Folge von Wörtern (Satzstruktur)) dieser „Fehler“ von nun an seltener passiert. Allerdings wird klar, dass eine KI unabdingbar Fehler machen muss, denn nur so kann diese lernen. Dies gilt es stets zu berücksichtigen, wodurch theoretische gesehen immer mehrere Permutationen betrachtet werden müssen, sollten Probleme entstehen, in welchen ein Input mehrere Ausgaben erzeugen könnte. Angemerkt sei zu dem Vorherigen, dass bekannte und bereits rein in den menschlichen Sprachen auftretende Problem (bzw. mögliche Missverständnisse bereits im Verstehen  ner Sprache, ausgelöst durch Mehrdeutigkeiten von Wörtern) in dieser Arbeit nicht näher verfolgt werden.

Aufgrund bekannter Technologien, wie z.B. Google Translate, DeepL und co. sollten solche Fehler innerhalb einzelner Wörter als ein „gelöstes Problem“ betrachtbar sein. Sprachliche Missverständnisse könnten aber immer dann entstehen, wenn sich mehrere Sprachen miteinander vermischen, welche sich ein Lexikon teilen (bspw. hier: $\text{T}_{\text{E}}\text{X}$, \LaTeX , ... und die, zunächst, englische Sprache). Die Frage ob ein Wort übersetzt werden darf oder nicht, unterscheidet einzelne Problemarten (Fälle). Die Hoffnung besteht, dass diese Probleme bereits gelöst sind, allerdings soll *ein Übersetzer* in der folgenden Schilderung alle Fehler machen *dürfen*.

2 Problemfälle

2.1 Herangehensweise

Reihenfolge

Die nachfolgende Auflistung verschiedener Fälle, welche Probleme gegenüber der $\text{T}_{\text{E}}\text{X}$ -Syntax, bzw. innerhalb von $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ -Dokumenten hervorrufen könnten, benötigt per se keine Reihenfolge, da sie möglichst alle unabhängig voneinander behoben sein sollen. Ein unbedachtes, zufälliges sequenzielles Nennen dieser könnte logische Lücken produzieren und damit ein Übersehen potentieller Fehler riskieren. Deshalb wird eine Reihenfolge gewählt, welche nicht auf $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ -Ebene beginnt, sondern sich so weit wie möglich dem Ursprung dieses Systemes nähert. Von den bereits in $\text{T}_{\text{E}}\text{X}$ auftretenden Problemen muss ein Weg in Richtung der auf dieser Software aufbauenden Technologien gebahnt werden. Da es sich der Gesamtheit der Quelltextdateien, auf welche ein Kompilervorgang von $\text{T}_{\text{E}}\text{X}$ zugreifen kann, immer um reine Textdateien handelt, werden spätere Beispiele nicht nach einzelnen Technologien betrachtet, sondern nach ihren Use-Cases (als relevante Systeme kämen hier zunächst $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$, $\text{BibT}_{\text{E}}\text{X}$, TikZ und Weiterführende in Frage, welche teils andere Probleme nach sich ziehen). Eine Fehlerunterscheidung findet nach Funktionalität in einem Dokument (das nach dem Kompilieren entstehende, ausgehend von einer Technologie, beschrieben in 2.1) und dessen virtuellem Pendant (mit Inhalten abhängig von anderen Dateien und Technologien, beschrieben in 2.1). In einem einzelnen Beispiel wird zunächst von dem strukturellen Element ein Beispiel abgelitten und danach eine abstraktere Beschreibung dieser Fehlerquelle gefunden.

Logische Strukturen in Dokumenten

Beispiele in reellen Dokumenten sind nach den Strukturen sortiert, welche man in Dokumenten beliebiger Natur wiederfinden kann. Als kleinste Struktur würde man hier (abgesehen von einzelnen Worten und Sätzen) Paragraphen sehen, welche Abschnitte eines Dokumentes formen. Mehrere dieser Abschnitte ergeben einen größeren Abschnitt. Statt von einem „Überabschnitt“ zu sprechen, wird daher die Formulierung umgekehrt. Ein Dokument ist zu Beginn als ein großer Abschnitt zu verstehen, welcher sich in verschiedene Unterabschnitte teilt, deren Namensgebung individuell sein kann. Hier wird (ähnlich wie bei: Knuth (1986)) von Kapiteln, Abschnitten, Unterabschnitten und Paragraphen gesprochen, jedoch mögliche „Unterunterabschnitte“ nicht als einzelne logische Struktur betrachtet, sondern als Unterabschnitt eines Unterabschnittes. Abschnitte selbst stellen aber nicht die einzige logische Struktur in einem Dokument dar, sondern es existieren auch andere, nicht-textliche Inhalte (die dem Kompilieren eines $\text{T}_{\text{E}}\text{X}$ -Quellcodes das Aussehen des Dokumentes beeinflussen). Die grösste Struktur eines Dokumentes entscheidet die Präambel von $\text{T}_{\text{E}}\text{X}$ -Dokumenten, welche sich aus verschiedenen Formen von Befehlen zusammensetzt, von welchen nur begrenzt viele Inhalte übersetzt werden dürften (meist: keine). Nach der Präambel, also im Teil, der *tatsächliche* Inhalte im Dokument beschreiben soll, ist nach Befehlen (Kommandos) und Umgebungen zu klassifizieren, deren Funktion innerhalb des Quellcodes auch nach einem Übersetzen erhalten bleiben muss. Im Normalfall sei davon auszugehen, dass Kommandos innerhalb eines Dokumentes entweder bestimmte globale Parameter definieren (bspw. in der Präambel) oder in andere graphische Elemente (bspw. Formelzeichen, Stilisierung von Zeichnungen, ...) aufgelöst werden und Umgebungen Änderungen an größeren Teilen eines Abschnittes vornehmen. Einzelne Elemente der Präambel deuten jedoch darauf hin, welche Art von anderen Technologien (auf $\text{T}_{\text{E}}\text{X}$ -aufbauend) in einem Dokument genutzt werden und wohingegen ein

übersetzendes Programm die Suche nach zu übersetzenden Strings über das eigentliche Dokument (bzw. einen einzelnen Quellcode) hinaus ausweiten muss.

Ein Dokument (T_EX oder L^AT_EX) kann/muss nicht nur mittels einem Quelltext beschrieben werden oder nur unter Nutzung eines T_EX-Compilers entstehen. Den eigentlichen (im Sinne: existierenden, mit Namen versehenen) Technologien abgesehen, müssen Dokumente teils aus Gründen der Lesbarkeit (des Quelltextes) oder logischen Reihenfolge von strukturellen Elementen Informationen in mehrere Dateien auslagern oder auf mehrere Dateien zugreifen, um gesamte Inhalte eines Dokumentes zu produzieren. Insbesondere die Lesbarkeit von Quellcodes kann von einer solchen Nutzung von mehreren Quelltextdateien für ein L^AT_EX-Dokument profitieren, da z.B. Graphiken (via TikZ o.Ä. erstellt) oder Quellcodes (mittels bspw. minted oder lstlistings dargestellt) ansonsten Quelltexte (für das T_EX-Dokument) produzieren können, die abertausende Zeilen lang sind. Andererseits kann (aus Sicht des T_EX-Parsers) keine Struktur in einem Dokument (ob übersetzt, ob unübersetzt) in einem bspw. Inhaltsverzeichnis gelistet sein, welches nach diesem Verzeichnis steht. Der Parser kann diese Struktur (beim Abarbeiten des Quelltextes von oben nach unten) nur sehen, nachdem er bereits den Befehl sah ein Inhaltsverzeichnis auszugeben. Die Informationen über die Inhalte dieses Inhaltsverzeichnis' erhält er logischerweise allerdings erst, nachdem er alle Teile des Dokumentes (dessen Beschreibung) gesehen hatte (und demnach in eine helfende Datei schreiben konnte, auf welche er beim nächsten Durchlaufen zugreifen kann).

Zudem sind ein paar zusätzliche, sprachliche und teils unlösbare Probleme gelistet, welche nicht unbedingt als Anforderungen der gegebenen Problemstellung zu verstehen sind und daher als „abweichend“ zu verstehen sind, aus welchen sich aber spätere Erweiterungspotentiale zeigen könnten.

Struktur eines Beispiels

Die Darstellung einzelner Beispiele erfolgt tabellarisch und demonstriert erst richtiges (bzw. zulässige) Verhalten und danach Fehlerhaftes (bzw. Unerwünschtes). Untiges Beispiel (Tabelle 1) dient hierbei als einfaches Beispiel für Fehler, die sich bei einem imaginären Befehl ink zeigen könnten. Dieser soll einen String mit einer bestimmten Farbe hinterlegen und besitzt einen zusätzlichen optionalen Farbparameter. „Richtig“ wäre es im originalen String nur das Wort in den geschwungenen Klammern zu übersetzen, da hierbei an keiner Stelle Information verloren geht und das Wort, nachdem es vom Deutschen ins Englische übersetzt wurde, weiterhin so wie vorgesehen hervorgehoben wird. „Zulässige“ Übersetzungen treten dann auf, wenn nur für die Formatierung (insofern hieraus keine weiteren Probleme entstehen) verloren geht. Im gegebenen Beispiel würde dann zwar die farbige Hinterlegung verloren gehen, das Wort allerdings trotzdem übersetzt werden und würde den Weg in ein Dokument finden, ohne einen sprachlichen Informationsverlust zu riskieren (für den Endnutzer/Leser).¹ „Unerwünscht“ sind Fälle, in denen ein Übersetzen Fehler für die T_EX-Engine produziert. Übersetzt man hier z.B. ink nach Tinte könnte es sich bei Zweiterem wiederum um einen anderen Befehl handeln, das Wort *Wort* einliest, aber eigentlich den alphanumerischen Wert von *word* erwartet hätte.² Ein Fehlschlagen des Befehls Tinte würde zwar einen Fehler für den T_EX-Parser produzieren, dieser wüsste dann aber, dass dieser Befehl bereits einmal fehlgeschlagen ist und eine neues Kompilieren verlangen, in welchem dieser Befehl und seine Optionen ignoriert werden, wodurch das Wort auch hier im Dokument landen würde. Man kann allerdings nicht bei

¹Selbst bei weißer Schriftfarbe kann das Wort in einem PDF-Reader markiert und kopiert werden.

² $57_{16} + 6f_{16} + 72_{16} + 74_{16} = 25 \times 16^1 + 28 = 400$ statt: $77_{16} + 6f_{16} + 72_{16} + 64_{16} = 26 \times 16^1 + 28 = 416$. Wofür der Befehl Tinte einen/den Integer 416 benötigt, kann ich Ihnen allerdings nicht erläutern.

jedem beliebigen T_EX-Befehl davon ausgehen, dass dieses Verhalten einheitlich auftreten wird. Hierbei existieren Fälle, welche dafür sorgen könnten, dass andere Wörter nun nicht mehr Teil eines Dokumentes werden könnten ?? „Fehlerhaftes“ Verhalten beim Übersetzen von T_EX-Quelltextdateien führt zu einem Informationsverlust, da das zu übersetzende Wort entweder nicht mehr übersetzt wird oder nicht mehr im Dokument wiederzufinden ist. Sobald man beginnt mit mehreren Dateien ein einziges Dokument zu beschreiben, riskiert ein naïves Übersetzen nur von einem Quelltext ausgehend, dass aus unerwünschten Fehlern innerhalb von einem Dokument fehlerhaftes Verhalten für das entstehende Produkt (meint: die kompilierte PDF) entsteht.

Abstrahiert man von diesem detaillierterem Beispiel, so sind „richtige“ Übersetzungen frei von Informationsverlust, „zulässige“ Übersetzungen nur dazu fähig Informationen für die graphische Aufbereitung (allerdings nicht den sprachlichen Inhalten) zu entwenden, „unerwünschte“ Übersetzungen dazu in Lage Informationen verbergen können und „falsche“ Übersetzungen fehlende sprachliche Inhalte innerhalb eines Dokumentes, sowie fehlende Übersetzungen dieser. Nicht jede Gruppe von Beispielen führt dazu, dass alle benannten Kategorien auftreten.

2.2 Elemente in einem T_EX-Quellcode

2.2.1 Die Präambel

Der unsichtbare Header eines T_EX-Dokumentes zeigt an (zunächst) wenigen, sehr spezifischen Stellen eine Schwierigkeit auf. In dieser sind meistens Informationen enthalten, welche nicht zu übersetzen sind, da sie z.B. Parameter für dokumentenweite (globale) Einstellungen setzen. Die Art und Weise *diese* zu setzen ist allerdings genauso behandelbar, wie einige folgende Kommandos (in Abschnitt ??) und wäre daher eigentlich nicht von weiterem Interesse, zeigt sich allerdings sehr geeignet dazu, einige triviale Fehlerquellen abzudecken (bzw. ungeeignete Ansätze). So kann man sich nicht immer gewiss sein, man einzelne Zeilen gesondert auswerten kann, da z.B. Pakete wie hyperref es in ihren Optionen (hypersetup) erlauben einige Einstellungen durch Zeilenbrüche voneinander getrennt (und übersichtlicher) darzustellen. Genauso sind allerdings auch einzelne Zeilen nicht direkt als „insgesamt T_EX syntaktisch“ zu betrachten, nur weil ein syntaktisches Element von T_EX innerhalb dieses vorliegt.

2.2.2 Kommandos

Kommandos selbst sind in der reinen T_EX-Syntax durch ein Backslash \ gekennzeichnet. Die Frage, ob das Wort, das einem Kommando folgt, übersetzt werden sollte, oder nicht, bringt Konflikte. Beispielsweise müsste innerhalb von Auflistungen der nach einem \item folgende String übersetzt werden. Andererseits wäre in einer Definition mit \def vorerst davon abzusehen den folgenden String zu übersetzen, da dieser ein neues Makro/-Kommando/etc. definieren zu sucht. Demnach darf \def\hello nicht zu: \def\hallo werden (es sei denn alle folgenden \hello werden auch übersetzt).³

Innerhalb des Quelltextes können also Strukturen auftreten, welche direkt folgende Zeichenketten als syntaktisches Element zur Dokumentenbeschreibung benötigen. Da hierbei beide Fälle auftreten könnten und nicht an dem Kommando selbst erkennbar sind, muss hierbei nach Art des Kommandos selbst abgewogen werden. Demnach genügt es nicht nur reine Befehle anhand des \ zu erkennen oder als festes Maß dafür zu nehmen, ob ein folgender String (der kein weiteres Muster aufweist) zu übersetzen ist, oder nicht.

Klammern für Parameter Auf viele Kommandos (Makros) folgen ein Paar geschwungener Klammern. Die Inhalte dieser sind in einigen Fällen zu übersetzen und in Anderen nicht, aber insgesamt als eine Art Parameter für dieses Kommando zu verstehen.

Beispielsweise wäre das Kommando \paragraph{This is a string of many characters} eines, in welchem die Inhalte der Klammern übersetzt werden sollen und das Kommando \usepackage{geometry} ein Beispiel, in welchem das Übersetzen von geometry zu Geometrie dazu führen würde, dass das entsprechende Paket nicht im Quelltext verwendet wird. Diese Art von Fällen, welche der Übersetzung ausgeschlossen werden soll, ist besonders kritisch, wenn mit mehreren Dateien gearbeitet wird und so würde ein Übersetzen von z.B. \include{clock.tex} (oder \include{clock}) zu \include{Uhr.tex} (oder: \include{Uhr}) dazu führen, dass größere Teile eines Dokumentes komplett ausgeschlossen werden.

³Da aber, wie bereits bekannt, immer eine Wahrscheinlichkeit, so gering sie auch sein mag, besteht, dass an einer nicht vorgesehenen Stelle eine Übersetzung auftreten *könnte*, kann man sich nicht auf die vorherige Aussage verlassen

Klammern dieser Art genügen also alleine noch nicht dafür eine Aussage darüber zu treffen, ob innenliegende Strings übersetzt werden dürfen, allerdings diese Zeichen (meint: die geschwungenen Klammern) kein Teil direkter Teil einer menschlichen Sprache und man würde davon meist erwarten, dass diese Klammern im Normalfall ein Indikator dafür sind, dass nicht übersetzt werden soll und in bestimmbar Ausnahmefällen (bspw. `\section{}`, `\paragraph*{}`, `\chapter{}`, `\title{}`, etc.) von Interesse für ein Übersetzen sind.

2.2.3 Optionen

Details: Optionen sind, neben Parametern, eine weitere Möglichkeit einem Kommando in T_EX zusätzliche Informationen mitzuliefern, z.B. für zusätzliche Formatierung. Aus logischer Reihenfolge müssten diese Optionen *eigentlich* immer vor der Zeichenkette, welche formatiert werden soll, stehen, da erst nach Einlesen der Parameter die Information dieser erkannt und auf den folgenden String angewendet werden kann. Diese Aussage wäre logisch, gilt allerdings nicht einheitlich (bspw. bei der Verwendung von TikZ-Bibliotheken). Optionen in eckigen Klammern insgesamt einer Übersetzung zu entziehen, verhindert allerdings die Möglichkeit, dass die Optionen einiger Befehle selbst ein String sein könnten, welcher im Dokument gedruckt und somit übersetzt werden muss.

Unvorhersagbarkeiten Eigene, definierte Makros erlauben für eine quasi-beliebige Zahl an Stellen, innerhalb welcher Strings erwartet werden könnten. So könnte beispielsweise ein Befehl definiert werden, welcher 7 Eingabeparameter erwartet: `\def\whatev a#1a#2a#3a#4a#5a#6a#7{This #1 is #2 to #3 be #4 troublesome. Hence we will` bei einem Aufruf der Form `\whatev amacrogoingtoaverya1a0a0` erwarten, dass sowohl alle menschen-sprachlichen Zeichenketten innerhalb der Definition (*This is to be troublesome. Hence we will only print one hundred now*), als auch außerhalb dieser (*macro going to very*) übersetzt werden und einen schlüssigen Satz bilden.

Strukturen


Umgebungen und Makros Große Strukturen in der T_EX-Syntax zeigen sich oftmals in sog. Umgebungen auf und stellen gerne/oft Graphiken `\var` (wie bspw. in TikZ), müssen dies allerdings nicht zwingend. Genauso wäre zu erwarten, dass größere Texte innerhalb dieser Umgebungen reinen Text-Formatierungen obliegen und damit gänzlich zu übersetzen wären. Allerdings können verschiedene Umgebungen auch eigene Syntaktik tragen (bspw. Tabellen), wodurch sich innerhalb von solchen Umgebungen sowohl zu übersetzende Strings, als auch zu Erhaltende (= nicht zu übersetzende Strings) verbergen können.

Vordefinierte


Insbesondere Problematisch wird die Ermittlung solcher bei Umgebungen, welche nicht mittels entsprechender `begin` und `end` Kommandos betreten/verlassen werden, sondern auf welche mittels eigener Symbolik ein access gewährt werden kann. Innerhalb reinen T_EX's sind diese an wenige Zeichenketten beschränkt und so können nur ein/zwei Dollar-Zeichen ($\$, \$\$$) oder `\(, \)`, `\[, \]` auf den/das Beginn/Ende einer ein-/ mehrzeiligen mathematischen Umgebung hindeuten. Darüberhinaus erlauben gerade diese Umgebungen den Wechsel in *normale* Umgebungen (also hinein in die Textumgebung des eigentlichen Dokumentes, aus welcher solche mathematische Umgebungen zu fliehen suchten) innerhalb welcher wieder in beschriebene mathematische Umgebungen gewechselt werden kann. Solange auf jeden Wechsel *in* eine solche Umgebung ein Wechsel *aus* einer solchen Umgebung heraus erfolgt, kann T_EX dies interpretieren und alle rein textlichen Strings *sollten* übersetzt werden.

Tabellen und Formeln Tabellarische Strukturen eignen sich für eine übersichtliche Darstellung von z.B. Messwerten oder auch mathematischer Formeln. Zwar ist bei z.B.

$$\begin{aligned}\sin(\omega t - k\vec{r}) &= 0 \\ \arcsin(\sin(\omega t - k\vec{r})) &= \arcsin(0) \\ (\omega t - k\vec{r}) &= 0 \\ \omega t = k\vec{r} \text{ note that: } \omega &= 2\pi f(\text{frequency})\end{aligned}$$

keine „gewöhnliche“ Tabelle zu erkennen, allerdings  man sich einzelne Terme innerhalb vor und/oder nach einer Äquivalenzrelation wie Inhalte einer Zelle vorstellen. Gegebenes Beispiel erhält seine Struktur im T_EX-Quelltext durch eine eigene Syntax innerhalb solcher Umgebungen. Hierbei kann allerdings auch der Fall, wie zum Beispiel innerhalb einer Tabelle, Strings einzelner Zellen zu übersetzen sind, andere jedoch nicht. Bereits obiges Beispiel zeigt Wörter, welche interessant für einen Übersetzer sein sollten. Anders ist dies jedoch bei

Eigene Makros und Logik Unterer Beispielsatz ist eine Möglichkeit einen String (Hello to this world:) zu erzeugen. Jedoch ist mit dieser Definition auch der String: Goodbye for now in der Form `\appendstring{now}{Goodbye}{for}` produzierbar. Diesem Muster entsprechend könnten (theoretisch) unendlich viele eckige Klammern zu übersetzende Strings beinhalten. Dies wäre zunächst unproblematisch unter der Vorannahme, dass alle diese Strings zu übersetzen sind. Was allerdings, wenn eine Definition der Form `\newcommand{\addandappend}[3][1][2]{#1+#2 ist #3 #1+}` vorliegt, in welcher für #3 ein String (hier denkbar: gleich) erwartet wird? Solche Erwartungen sind nicht immer vorhersagbar und können dadurch nur schwer beschrieben werden (sind allerdings deterministisch nachvollziehbar, da die \TeX -Engine solche Makros schließlich auch interpretieren und rendern kann).

Problematisch wird dieser Fall, wenn solche Makros eigene und optionale Parameter in beliebiger Reihenfolge erwarten wollen, wie das folgende Beispiel (??). Übersetzt man z.B. das Wort „sunny“ auch nur an einer Stelle nicht, riskiert man Fälle, in welchen falsche Inhalte angezeigt werden. Auch kann man sich nicht gewiss sein, dass zu übersetzende Optionen oder Parameter immer in einheitlicher und vorhersagbarer Reihenfolge auftreten werden. 

Pakete Pakete sind ihrerseits keine, innerhalb eines Quelltextes direkt vorliegende Struktur, aber besitzen die Fähigkeit, dass sie Zeichenketten erwarten/tragen könnten, welche es zu übersetzen gilt. Zwar erfolgt die Einbindung solcher immer auf gleicherlei art und weise, aber hier einzelne befehle sind... übersetzbar... morgen anzupassen, weil ich kann nicht mehr

Klassen

English	Mögliche Übersetzung
Richtiges Verhalten	
<div> <code>1\ink[red]{word}</code> </div> <div> \TeX Code 1: Original </div>	<div> <code>1\ink[red]{Wort}</code> </div> <div> \TeX Code 2: Beispielübersetzung </div>
Zulässiges Verhalten	
<div> <code>1\ink[red]{word}</code> </div> <div> \TeX Code 3: Original </div>	<div> <code>1\ink[Rot]{Wort}</code> </div> <div> \TeX Code 4: Beispielübersetzung </div>
Unerwünschtes Verhalten	
<div> <code>1\ink[red]{word}</code> </div> <div> \TeX Code 5: Original </div>	<div> <code>1\Tinte[Rot]{Wort}</code> </div> <div> \TeX Code 6: Beispielübersetzung </div>
Falsches Verhalten	
<div> <code>1\ink[red]{word}</code> </div> <div> \TeX Code 7: Original </div>	<div> <code>1\Tinte[Rot]{word}</code> </div> <div> \TeX Code 8: Beispielübersetzung </div>

Tabelle 1: Abstrakte Struktur der folgenden Beispiele

```
1\begin{table}[h!tb]
2  \centering
3  \begin{tabularx}{\textwidth}{X X X}
4    \toprule
5      English words & Value of the first word & Value of the second word \\
6    \midrule
7      finite fields & 639 & 663 \\[-13px]
8      limited areas & 744 & 556 \\[1em]
9    \midrule
10     Galois field & 607 & 548 \\[-13px]
11     infinite set & 854 & 364 \\[1em]
12   \midrule
13     place & 517 & \\[-13px]
14     set & 332 & \\[-1em]
15   \bottomrule
16 \end{tabularx}
17 \caption{Several english strings and the summed up ASCII-values of their
18 ↪respective words.}
18\end{table}
```

TeX Code 9: Original

```
1\begin{table}[h!tb]
2  \centering
3  \begin{tabularx}{\textwidth}{X X X}
4    \toprule
5      English words & Value of the first word & Value of the second word \\
6    \midrule
7      finite fields & 639 & 663 \\[-13px]
8      limited areas & 744 & 556 \\[1em]
9    \midrule
10     Galois field & 607 & 548 \\[-13px]
11     infinite set & 854 & 364 \\[1em]
12   \midrule
13     place & 517 & \\[-13px]
14     set & 332 & \\[-1em]
15   \bottomrule
```

English Original

```
1\newcommand*{\appendstring}[3]{#2 #3 #1}  
2\appendstring{world:}{Hello}{to this}
```

TeX Code 11: Original

Ideale Übersetzung

```
1\newcommand*{\appendstring}[3]{#2 #3 #1}  
2\appendstring{Welt:}{Hallo}{an diese}
```

TeX Code 12: Beispielübersetzung

Tabelle 3: Beispiel für die Fähigkeit, dass Makros einzelne Strings einlesen können

```
1\newcommand{\weather}[2][sunny]
2{
3  Heute war es
4  \ifthenelse{\equal{#1}{sunny}}
5  {sonnig.}
6  {nicht sonnig. Hoffentlich ist es morgen nach dem Regen wieder
7    \ifthenelse{\equal{#2}{rainy}}
8    {
9      regnerisch, da die Wälder zu trocken sind.
10   }
11   {
12     wieder sonnig werden.
13   }
14 };
15}
16
17\weather{sunny}\\\\
18\weather{rainy}\\\\
19\weather[rainy]{sunny}\\\\
20\weather{rainy}{rainy}\\\\
21\weather{rainy}{rainy}\\\\
```

TeX Code 13: Original

```
1\newcommand{\weather}[2][sonnig]
2{
3  Heute war es
4  \ifthenelse{\equal{#1}{sonnig}}
5  {sonnig.}
6  {nicht sonnig. Hoffentlich ist es morgen nach dem Regen wieder
7    \ifthenelse{\equal{#2}{regnerisch}}
8    {
9      regnerisch, da die Wälder zu trocken sind.
10   }
11   {
12     wieder sonnig werden.
13   }
14 }
```

3 Stand der Technik

3.1 Anforderungen

Abgelitten aus der Problemliste werden hier die Probleme umformuliert als Anforderungen dargestellt und in absteigender Reihenfolge nach Relevanz in Bezug auf die gegebene Aufgabenstellung aufgeführt.

Die Technologien dienen den Anforderungen, sollten sie:

1. kompilierbare Dokumente erzeugen
2. alle Abschnitte in Dokumenten übersetzen
3. kontextuell terminologisch richtige Übersetzungen wählen (die richtigen Lexeme/Wörter treffen)
4. den Kontext selbstständig aus den wörtlichen und erreichbaren (lokalen) Informationen (Dateien) ablesen können
5. den Kontext aus den mathematischen, graphischen, tabellarischen, ... Inhalten einer Datei ablesen können
6. den Kontext aus externen Verweisen (Links) erfassen können (Lokal, als auch Web)
7. ...

3.2 Denkbare Ansätze

Alle Lösungswege und Workflows, die ich mir vorstellen kann und denken konnte. Definiert evtl. Rollen,

3.3 Existierende Ansätze

Alle Technologien, die diese Rolle (n) in den entsprechenden Ansätzen füllen könnten.

3.3.1 Testverfahren

logischerweise: In den denkbaren Ansätzen schon gegenargumentieren, was unsinnig ist und warum. Reduziert die Menge an zu testenden Lösungen.

3.3.2 Durchführung

3.3.3 Auswertung

3.4 Grenzen der Lösungen

3.5 Takeaways

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

Literatur

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

Lamport, L. (1994), *LaTeX: A Document Preparation System, 2nd Edition*, ISBN: 9780201529838, Addison-Wesley Professional. available at: <https://www.latex-project.org/help/books/tlc3-digital-chapter-samples.pdf> (last Access: 04.10.2025).

Shannon, C. E. (1948), 'The mathematical theory of communication', *The Bell System Technical Journal*, ISSN: 0343-6993 (vol. 27). Harvard Reprint available at <https://people.math.harvard.edu/~ctm/home/text/others/shannon/entropy/entropy.pdf> (last Access: 16.10.2025).

A Anhänge

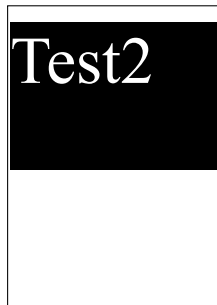
A.1 Fontskalierung auf Webseiten

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 $\text{T}_{\text{E}}\text{X}$ geschehen, sollte

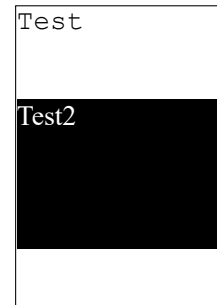
```
<html>
  <head>
    <title>Example</title>
    <style>
      /*formatting options are: none and black*/
      .t{
        font-size:13em;
        height:50%;
      }
      /*formatting option: none = no background, black, courier*/
      .t#none{
        font-family: 'Courier New', Courier, monospace;
      }
      /*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 TikZ 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