

Lilly is a Latex Lovable Yogurt

— It doesn't have to make any sense if it looks beautiful —

Dokumentation – Version 1.0.10

Autor & Instandhaltung:

Florian Sihler (florian.sihler@web.de)

15. Juli 2019

Abstract

Oder auch Einleitung ❤️ für **VER 1.0.10**

Die L^AT_EX-Dokumentklasse **Lilly** ist im Rahmen des Studiums von Florian Sihler entstanden, und dient der Generierung studiumsrelevanter Dokumente & Mitschriften, in dessen Rahmen Lilly weiter angepasst und (hoffentlich) optimiert wurde. Die klassische Version basiert auf der [KOMA-Script](#) Dokumentklasse scrbook.

Das Ziel ist es auf Basis eines Makefiles das Latexdokument direkt in verschiedenen Versionen zu generieren! Die aktuelle Version „1.0.10 - Jake ist auch nur Java“ besitzt den Status Work in Progress!

Inhaltsverzeichnis

1 Einleitung	1
1.1 Installieren von Lilly	1
1.1.1 Linux	
1.1.2 Windows	
1.1.3 MacOS	
1.1.4 Keine Installation	
1.2 Erstellen eines Dokuments mit Lilly VER 1.0.5	3
1.2.1 Das Gerüst	
1.2.2 Die Böxli	
1.2.3 Hyperlinks	
1.3 Einbinden von weiteren Dokumenten	5
1.3.1 Aufgliedern eines Dokuments	
1.3.2 Übungsblätter	
2 Mathe	8
2.1 Weitere Befehle	8
2.1.1 Operatoren	
2.1.2 Symbole	
2.1.3 Kompatibilität	
2.2 Plots VER 1.0.8	10
2.2.1 graph-Environment	
2.2.2 wgraph-Environment	
2.3 3D-Plots WAR Ausstehend	11
3 Grafiken	12
3.1 Grundlegende Symboliken	12
3.1.1 Die Ampeln	
3.1.2 Emoticons WAR Ausstehend	
3.1.3 Utility WAR Ausstehend	
3.1.4 Titleimages WAR Ausstehend	
3.2 Diagramme & Graphen	13
3.2.1 Graphen	
3.2.2 Rotation	
3.2.3 Automaten WAR Work in Progress	
3.2.4 Schaltkreise WAR Ausstehend	
3.2.5 Neuronen WAR Work in Progress	
4 Farben	17
4.1 Die normalen Farbprofile	17
4.1.1 Das Standardfarbprofil	

4.1.2 Das Druckprofil	19
4.2 Weitere Planungen	19
5 Listings	20
5.1 Die grundlegenden Eigenschaften	20
5.1.1 Grundlegendes Design	
5.1.2 Das MAIN-Paket	
5.1.3 Das MIPS-Paket	
5.2 Die mitgelieferten Erweiterungen	22
5.2.1 assembler	
5.2.2 pseudo	
5.2.3 pseudo_nospace	WAR Veraltet
5.2.4 bash	
5.2.5 latex	
6 Boxen	24
6.1 Grundlegendes	24
6.1.1 Eine kleine Einführung	
6.1.2 Der Box-Controller	
6.2 Die Boxmodi	26
6.2.1 Default-Design	
7 Jake	28
7.1 Grundlegendes	28
7.1.1 Entwicklung	
7.1.2 Die Installation	
7.1.3 Die Schnittstelle	
8 Aussicht	29
8.1 Bekannte Probleme	29
8.1.1 TikZ	
8.2 Todos	29
8.2.1 Visuals	
8.2.2 Fehler	
8.2.3 Dateiaufteilung	
8.2.4 Road to CTAN	
8.2.5 Hoverover tooltips	
8.2.6 Weitere	
9 Anhang	31
9.1 Version VER 1.0.7	31
9.1.1 Installation in Linux	
9.1.2 Spezifikation: Plots	
9.2 Version VER 1.0.9	33
9.2.1 Installation in Linux	
9.2.2 Installation in MacOS	



EINLEITUNG

INTEGRIEREN VON LILLY – DIE GRUNDLAGEN VON A-Z


1.1 Installieren von Lilly



Aktuell kommt die Dokumentklasse ohne `.ins` oder `.dtx` Datei, dafür allerdings mit einem Installer für alle Debian (Linux) basierten Betriebssysteme, an einer Variante für MacOS und Windows wird momentan entwickelt.

VER 1.0.10

Bemerkung 1.1 – Mithilfe

Wenn du dich mit $\text{T}_{\text{E}}\text{X}$ oder $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ auskennst, schreibe an folgende Email-Adresse florian.sihler@web.de.

Mittlerweile gibt es auch ein offizielles Github-Repository (<https://github.com/EagleoutIce/LILLY> ) über das die gesamte Entwicklung abläuft. Hier werden noch Helfer für folgende Aufgaben gesucht:

- | | |
|--|---|
| ◇ Java - Entwicklung | ◇ Kommentieren in Doxygen |
| ◇ Bash, Konsolen - Entwicklung | ◇ Layout Gestaltung |
| ◇ Kommentieren in Markdown | ◇ $\text{T}_{\text{E}}\text{X}$, $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ -Entwicklung |
| ◇ Maintaining ($\text{T}_{\text{E}}\text{X}$, $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$) | ◇ Tester ( ,  , ) |

1.1.1 Linux

Für Versionen $< 1.0.10$ klicke hier: [klick mich!](#)

Mit der Portierung von *Jake* in die Programmiersprache Java hat sich die Installation von LILLY, immens vereinfacht. Da man hierfür allerdings *Jake* benötigt, der sich dann um alles weitere kümmert, sei hier einmal nur kurz erklärt, wie man die `stable`-Version von Jake installiert, für mehr Infos siehe: [Jake Installieren](#).

Mit dem Bezug dieser Dokumentation sollte eine `jake.jar` Datei einhergegangen sein, die es nun gilt auszuführen. Natürlich wird hierfür Java benötigt, auf einem apt-Basierten Betriebssystem installiert man Java wie folgt:

```
1 sudo apt install default-jdk
```

Für alle anderen Derivate gilt es sich auf <https://www.oracle.com/de/java/> entsprechend zu informieren. Einmal installiert, genügt ein Ausführen der `.jar` Datei mithilfe von `java -jar jake.jar` oder durch einen Doppelklick, sofern die entsprechende `.jar` als Ausführbar markiert ist. Zieht man bunte Fenster der Kommandozeile vor, so ist man in der Lage mit

`java -jar jake.jar GUI` eine grafische Unterstützung zu erhalten, die allerdings momentan noch in Arbeit und noch lange davon entfernt ist, dieselbe Mächtigkeit wie die Kommandozeile zu erreichen.

Lilly mit Jake installieren

Nun genügt ein Ausführen von `jake install`, wobei mithilfe der Option `-lilly-path` der Pfad angegeben werden kann, an dem sich die LILLY.cls befindet:

```
1 jake install -lilly-path: '/absoluter/Pfad/zum/Lilly/Ordner'
```

Anschließend sollte es möglich sein Dokumente mit LILLY zu kompilieren. Gemeinsam mit LILLY werden eine Vielzahl an Beispieldokumenten ausgeliefert, die die Verwendung anschaulich machen sollen und somit auch als Test für eine erfolgreiche Installation verwendet werden können. Exemplarisch sei `test & bonus/map_tests/test.conf` genannt, welches auch die `getGraphics`-Schnittstelle etabliert.

1.1.2 Windows

1.1.3 MacOS

1.1.4 Keine Installation

Bemerkung 1.2

Von dieser Methode wird abgeraten

Natürlich lässt sich Lilly auch so nutzen, hierfür muss einfach nur die zu kompilierende Latex-Datei im selben Verzeichnis wie die Datei `Lilly.cls` liegen (also: `Lilly`). Natürlich kann dies bei mehreren Dateien, die auf Lilly zugreifen, unübersichtlich werden.

1.2 Erstellen eines Dokuments mit Lilly VER 1.0.5

1.2.1 Das Gerüst

Es ist recht einfach ein Dokument mit Lilly zu erstellen. Da es sich ja um eine Dokumentenklasse handelt, wird sie wie folgt eingebunden:

```
1 \documentclass[Typ=Dokumentation]{Lilly}
```

Für den Typ gibt es hierfür 4 Optionen:

VER 1.0.7

- ◇ Dokumentation
- ◇ Uebungsblatt
- ◇ Mitschrieb
- ◇ Zusammenfassung

Mit VER 1.0.10 ist es möglich auch nur `Dokumentation` anstelle von `Typ=Dokumentation` zu schreiben. Die Definition für dieses Dokument lautet zum Beispiel:

```
1 \documentclass[Dokumentation]{Lilly}
```

In Kombination mit *Jake* ist es zudem noch möglich die Option `Jake` anzugeben, die es Jake gestattet die Dokumentspezifischen Parameter zu bestimmen.

Zu beachten ist, dass die anderen Optionen weitere Parameter *fordern*.

So benötigt Mitschrieb noch den Parameter `Vorlesung`, der zusammen mit dem Parameter `Semester` gemäß:

Semester ist Standardmäßig 1

```
1 \input{\LILLYxPATHxDATA/Semester/\LILLYxSemester/Definitions/
2 \LILLYxVorlesung}
```

die für die jeweilige Vorlesung definierten Daten lädt. Erklärungen für die geladenen Daten befinden sich in den jeweiligen README-Dateien:

1. Semester `../Lilly/source/Data/Semester/1/Readme.md`

2. Semester `../Lilly/source/Data/Semester/2/Readme.md`

Weiter nutzt *Uebungsblatt* ebenfalls `Vorlesung&Semester` sowie noch die optionale Option (tihihi) `n` die angibt, um das wievielte Übungsblatt es sich handelt. Darüber müssen wir uns aber in der Regel keine Gedanken machen. Trägt unser Übungsblatt einen Namen wie `uebungablatt-gdbs-42.tex`, so kann *Jake* über sogenannte NameMaps entsprechend alles konfigurieren, in diesem Fall benötigt dein Übungsblatt auch kein `documentclass` mehr, es genügt das direkte Schreiben von Latex-Code, der Rest wird von Jake übernommen.

Entsprechend des Dokumenttyps werden gegebenenfalls auch bereits etliche Seiten generiert, dies gilt es zu beachten, wenn man vielleicht nur etwas testen möchte. In diesem Fall gibt es (wie später auch noch weiter aufgeführt) den sogenannten *Bonustyp* PLAIN, welcher ein leeres Dokument erstellt!

1.2.2 Die Böxli

Jede Box besteht als Environment und lässt sich wie folgt nutzen:

Definition 1 – Titel



Hallo Welt

```
1 \begin{definition*}[Titel]
2   Hallo welt
3 \end{definition*}
```

Satz 1.1 – Titel

Hallo Welt

```
1 \begin{satz}[Titel]
2   Hallo welt
3 \end{satz}
```

Definition 2

Hallo Welt

```
1 \begin{definition}
2   Hallo welt
3 \end{definition}
```

Aufgabe 0.1 – Titel

3 Punkte

Hallo Welt

```
1 \begin{aufgabe}[Titel][3]
2   Hallo welt
3 \end{aufgabe}
```

Letztere ändert sich zum Beispiel mit dem Dokumenttyp, so wird die Aufgabenbox in einem Übungsblatt immernoch wie folgt veranschaulicht:

Aufgabe 2 – Titel

Hallo Welt

```
1 \begin{aufgabe}[Titel][3]
2   Hallo welt
3 \end{aufgabe}
```

Hier eine Liste aller Boxen:

- | | | |
|--------------|----------|-------------------|
| ◇ definition | ◇ satz | ◇ zusammenfassung |
| ◇ bemerkung | ◇ beweis | ◇ aufgabe |
| ◇ beispiel | ◇ lemma | ◇ uebungsblatt |

Sie können alle mithilfe von:

```
1 %% Allgemein
2 % \LILLYcommand{\LILLYxBOXx<FirstLetterUp-Name>xEnable}{FALSE}
3 \LILLYcommand{\LILLYxBOXxDefinitionxEnable}{FALSE}
```

jeweils deaktiviert und damit aus dem Dokument entfernt werden (auch nur abschnittsweise, das Reaktivieren funktioniert analog mit TRUE).

Eine Auflistung ihrer lässt sich mit dem `\listof` Befehl erzeugen. Beispielhaft:

```
1 \listofDEFINITIONS
```

erzeugt hierbei:

Die Bezeichnung der Listen sind bisher noch inkonsistent :/

Natürlich sind die Linien nur zur Trennung eingefügt

Alle Definitionen

1	📌 Titel	4
2		4
3	Titel	24
4	📌 Titel	25

1.2.3 Hyperlinks

Eine Sprungmarke innerhalb eines Dokuments lässt sich mit:

VER 1.0.0

```
1 \elable{mrk:Hey} %% \elable{<Sprungmarke>}
```

erstellen. Referenziert werden kann sie mithilfe des Befehls **jmark**:

```
1 \jmark[Klick mich]{mrk:Hey} %% \jmark[Text]{Sprungmarke}
```

der erzeugte Link: **Klick mich**, passt sich zudem der Akzentfarbe der aktuellen Boxumgebung und dem Druckmodus an:

Zusammenfassung 1.1 – Testzusammenfassung

Siehe hier: **Klick mich** (Wenn Druck: Klick mich → ⁵)

Der alternative Vertreter für **jmark** ist **hmark**, er ignoriert sämtliche Farbattribute:

```
1 \hmark[Klick mich]{mrk:Hey} %% \hmark[Text]{Sprungmarke}
```

und erzeugt damit: **Klick mich**.

jmark*, welcher die Akzentfarbe ignoriert, ist bereits geplant, wurde allerdings bisher nicht zwangsläufig benötigt

1.3 Einbinden von weiteren Dokumenten

1.3.1 Aufgliedern eines Dokuments

Um Dokumente portabel kompilierbar zu machen, setzt das Makefile gemäß der Konfiguration **\LILLYxPATH** (hier: „./“). Nun lässt sich mithilfe des Befehls **\linput{<Pfad>}** eine Datei relativ zur Quelldatei angeben (beachte, dass absolute Pfade bei **\linput** keinen Sinn machen. Hierfür solltest du weiterhin **\input** verwenden).

Zudem lässt sich damit über **\LILLYxDOCUMENTxSUBNAME** der Name der zuletzt eingebundenen Datei (Data/Einleitung.doc) abfragen.

Weiter gilt zu beachten, dass es *nicht* möglich ist, das klassische **\include** zu verwenden! Dieser Befehl wird aber von LILLY deswegen direkt entsprechend erneuert (hierzu wird das klassische Latex **\input** im Zusammenspiel mit **\clearpage** verwendet, nicht LILLYs **\linput**!). Es ist also im Endeffekt doch möglich Dokumente mit **\include** zu verwenden.

VER 1.0.4

VER 1.0.7

1.3.2 Übungsblätter

Da es von Bedeutung ist Übungsblätter so zu erstellen, dass die Abgaben direkt in die Mitschrift eingebunden werden können, gibt es hierfür eine einfache Möglichkeit:

```

1 %% \inputUB{<Name>}{<Nummer>}{<Pfad - linput>}
2 \inputUB{Mengen}{1}{Aufgaben_Data/Uebungsblatt_1.tex}
3
4 %% Wird zu:
5 \clearpage
6 \begin{uebungsblatt}[Mengen][1]
7   \linput{Aufgaben_Data/Uebungsblatt_1.tex}
8 \end{uebungsblatt}
9 \newpage

```

Aktuell wird daran gearbeitet eine `make`-Regel für Übungsblätter zu integrieren

Die Verwendung von `\tcblower` ist noch in Arbeit

Es gibt auch eine Umgebung mit `*` und gleichermaßen `\inputUBS`. Diese setzen den Zähler für die Aufgaben *nicht* zurück!

Übungsblätter sind nur in **complete**-Varianten verfügbar, werden also sonst nicht eingebunden!

Ein Übungsblatt erstellen

Doch wie erstellt man nun ein Fachgerechtes Übungsblatt. Nun, da es sich hier um die Schnelleinführung handelt, ein paar Vorgaben. Benenne dein Übungsblatt nach dem Schema:

uebungsblatt-**<VORLESUNG>**-**<BLATTNUMMER>**.tex

Die Reihenfolge spielt keine Rolle, ein beispielhafter Name könnte sein: `gdb-s-uebungsblatt-13.tex`. Nun erstelle dir eine `jake.conf`-Datei, wobei egal ist wie sie heißt, solange sei auf `.conf` endet (fürs Autocomplete ☺). In sie trägst du folgendes ein:

```

1 file          = @[SELTEXF]
2 operation     = file_compile
3
4 lilly-modes   = uebungsblatt
5
6 lilly-show-boxname = false
7
8 lilly-nameprefix = FlorianS-Partner-
9 lilly-author   = Florian Sihler, Mein Partner
10
11 lilly-n       = @[AUTONUM]

```

Natürlich kannst du die Namen entsprechend ändern. Das sieht jetzt aus wie viel, aber das musst du nur einmal machen, sofern du die Konfigurationsdatei immer in das Verzeichnis mitkopierst, indem sich die Übungsblatt-`.tex` und **nur** diese `.tex`-Datei befindet. Wir werden uns später mit besseren Konfigurationen beschäftigen, die keinerlei Nachaufwand benötigen und galanter sind. In das Übungsblatt können wir nun unsere Aufgaben stecken. Hier ist der *gesamte* Inhalt der oben genannten `TEX`-Datei:

```

1 \begin{aufgabe}[Tolle Aufgabe][400] % 400 Punkte
2   Die Aufgabenbeschreibung, blah, blah, blah, \ldots
3   \begin{enumerate}
4     \item Teilaufgabe a)

```

```
5     \item Teilaufgabe b)
6     \item \ldots
7     \end{enumeratea}
8 \vSplitter
9     \begin{enumeratea}
10    \item Antwort zu Teilaufgabe a)
11    \item Antwort zu Teilaufgabe b)
12    \item \ldots
13    \end{enumeratea}
14 \end{aufgabe}
15
16 %% Weitere Aufgaben, wenn gewünscht
```

Kompilieren kann man den Spaß nun mit: `jake jake.conf`. Und das wars, Boom ☺.

MATHE

EINZELNE VARIATIONEN UND EINE MENGE ABKÜRZUNGEN

VER 1.0.0

An sich ändert LILLY nicht viel an der normalen Implementation der Matheumgebung. Die verwendete Matheumgebung lässt sich mithilfe des Befehls `\LILLYxMathxMode` frei einstellen. Standardmäßig wird dieser Wert auf *normal* gesetzt.

`\LILLYxMathxMode`

2.1 Weitere Befehle

2.1.1 Operatoren

Diese Definitionen befinden sich in der Datei: Maths/_LILLY_MATHS_OPERATORS

`\overbar` Lilly liefert den Befehl `\overbar{1}` dieser wird auf Basis von `mkern` so definiert, dass er direkt Abstände zwischen den Overlines definiert. So ergibt sich:

<code>\overbar{a_1} \overbar{a_2}</code>	$\overline{a_1} \overline{a_2}$
<code>\overline{a_1} \overline{a_2}</code>	$\overline{a_1 a_2}$

`\das` Für Definitionen gibt es die Befehle `\das` ($:=$), `\sad` ($=$), `\daseq` ($:\Leftrightarrow$), `\quesad` (\Leftrightarrow) sowie `\shouldeq` ($\stackrel{!}{=}$). All diese Befehle funktionieren lediglich in einer Matheumgebung und werden nicht mit `\ensuremath` abgesichert!

`\shouldeq` Bis auf den letzten werden zudem alle Befehle mithilfe von `\vcentcolon` realisiert.

`\sqrt` Weiter wurde das Aussehen der Wurzel verändert, so liefert nun der Befehl `\sqrt[1]{1}` folgendes:

<code>\sqrt[3]{42}</code>	$\sqrt[3]{42}$
<code>\oldsqrt[3]{42}</code>	$\sqrt[3]{42}$

Zudem gibt es einige Vereinfachungen für etliche typischen mathematischen Operatoren:

`\det` `\det` (\det), `\adj` (adj), `\LH` (\mathcal{LH}), `\eig` (Eig), `\Dim` (dim), `\sel` (SEL), `\sign` (sign), `\diag` (diag), `\LK` (LK), `\rg` (rg), `\KER` (ker), `\Eig` (Eig), `\cd` \cdot . Auch wurde das Aussehen von `\mod`, `\Im` und `\Re` modifiziert:

`\det`
`\adj`
`\LH`
`\eig`
`\Dim`
`\sel`
`\sign`
`\diag`
`\LK`
`\rg`
`\KER`
`\Eig`
`\cd`

MOD	\Im	\Re
<code>\mod</code>	<code>\Im</code>	<code>\Re</code>

Des Weiteren wurde noch die Matrixumgebung (`\env@matrix`) so erweitert, dass sie als optionales Argument eine gültige Array-Spaltendefinition entgegennimmt:

1	<code>\begin{pmatrix}[cc c]</code>	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$
2	<code>1 & 2 & 3 \\\</code>	
3	<code>4 & 5 & 6</code>	
4	<code>\end{pmatrix}</code>	

2.1.2 Symbole

Diese Definitionen befinden sich in der Datei: Maths/_LILLY_MATHS_SYMBOLS

Für die einzelnen Zahlenräume werden einige Befehle zur Verfügung gestellt, die alle über `\ensuremath` abgesichert sind: `\N` (\mathbb{N}), `\Z` (\mathbb{Z}), `\Q` (\mathbb{Q}), `\R` (\mathbb{R}), `\C` (\mathbb{C}). Sie werden mithilfe von `\mathbb` generiert. Auch die komplexe Einheit `i` wird mit `\i` zur Verfügung gestellt.

Weiter wurden die griechischen Buchstaben Epsilon und Phi modifiziert:

<code>\oldepsilon</code>	ϵ	<code>\epsilon</code>	ε
<code>\oldphi</code>	ϕ	<code>\phi</code>	φ

Zudem wird zum Beispiel die Menge der Binärzahlen über `\B` (\mathbb{B}), die Chromatische Zahl über `\X` (χ) und der generelle Körper mit `\K` (\mathbb{K}) zur Verfügung gestellt. Für die Potenzmenge liefert LILLY `\P` (\mathcal{P}), für die Menge der Funktionen `\F` (\mathcal{F}) und für die Groß-O-Notation `\O` (\mathcal{O}).

Weiter bindet LILLY das `pi font` Paket ein und liefert so zum Beispiel `\ding{51}` (✓) und `\ding{55}` (✗).

2.1.3 Kompatibilität

Diese Definitionen befinden sich in der Datei: Maths/_LILLY_MATHS_COMPATIBILITIES

Hier werden einige Befehle eingerichtet, die entweder noch nicht zugeordnet wurden oder während der Vorlesung (im Überlebenskampf :P) ins `eagleStudiPackage` eingebaut worden sind. Darunter vor allem für die *Lineare Algebra* kreierten: `\enum(1)` (`enumerate` mit `\narrowitems` (TODO: LINK)) und `\liste(1)` (`enumerate` mit römischen Zahlen und `\narrowitems`).

Weiter existieren die Befehle `\xa` ($\overline{x_1}$), `\xb` ($\overline{x_2}$), `\xc` ($\overline{x_3}$).

Für TikZ gibt es noch die Befehle `\crossAT(1)` (X^1) und analog `\circAT(1)` (O^2), sowie `\bblock(2)` ($\boxed{42}^3$). Diese sollen auf jedenfall noch in ein geeignetes TikZ-Dokument übertragen werden (TODO:!).

Weiter werden drei (mittlerweile obsolete) Umgebungen definiert:

- ◇ `nstabbing`: `tabbing`-Umgebung, ohne Abstände
- ◇ `centered`: `center`-Umgebung, ohne Abstände


¹`\tikz{\crossAT{(0,0)};}` – Zum Erhalt der Textzeile vertikal um -0.35baselineskip verschoben.

²`\tikz{\circAT{(0,0)};}`

³`\tikz{\bblock{(0,0)}{42};}` – Wieder vertikal um -0.2baselineskip verschoben.

◇ `squares`: Ähnelt `cases` - nur mit `'`'.

`\VRule` Zudem definiert sich noch für Tabellen der Befehl `\VRule[1]`, welcher eine Spalte variabler Größe für Tabellen zur Verfügung stellt. Eine exemplarische Einbindung findet sich hier:

<pre> 1 \begin{tabular}{c!{\VRule[6pt]}c} 2 \specialrule{2pt}{0pt}{0pt} 3 You're my & Wonder Wall\\ 4 \specialrule{2pt}{0pt}{0pt} 5 \end{tabular} </pre>	
--	--

Weiter gibt es noch einige verschiedene Tabellen-Spalten, deren Kurzbezeichner den Anschein erwecken wild zusammengewürfelt zu sein:

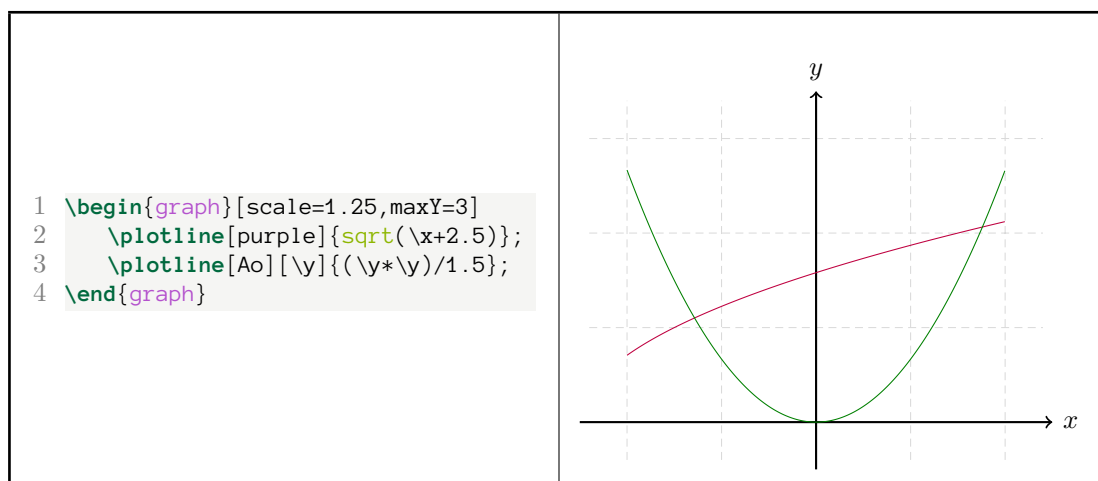
- ◇ `b`: Fettgedruckt zentriert
- ◇ `w`: Fußnotengröße linksbündig(X)
- ◇ `R(1)`: Forciert rechts mit Breite #1
- ◇ `u`: Mathematisch zentriert
- ◇ `L(1)`: Forciert links mit Breite #1
- ◇ `C(1)`: Forciert zentriert mit Breite #1
- ◇ `g`: Fußnotengröße linksbündig

2.2 Plots VER 1.0.8

Für die Spezifikationen siehe hier: [klick mich!](#)

2.2.1 graph-Environment

Es existiert die folgende Implementation der Graph-Umgebung:



Die Signatur des Environments lautet: `[Argumente][tikz-Argumente]`. Letzere sind einfach zusätzliche Argumente die der unterliegenden `tikzpicture` beigelegt werden. Für die Argumente gibt es folgende Möglichkeiten:

env: graph

Bezeichner	Typ	Standard	Beschreibung
scale	<i>Zahl</i>	1	Skalierungsfaktor
minX	<i>Zahl</i>	-2	X-Achse Start
maxX	<i>Zahl</i>	2	X-Achse Ende
minY	<i>Zahl</i>	0	Y-Achse Start
maxY	<i>Zahl</i>	4	Y-Achse Ende
offset	<i>Zahl</i>	0.4	Zusatzlänge Achsen
loffset	<i>Zahl</i>	0.1	Unbeachteter Zusatz Achsen
labelX	<i>String</i>	\$x\$	Bezeichner X-Achse
labelY	<i>String</i>	\$y\$	Bezeichner Y-Achse
samples	<i>Zahl</i>	250	Anzahl an Kalkulationen

2.2.2 wgraph-Environment

Um die Graph-Umgebung noch vielfältiger zu gestalten wurde geschaffen. Nach reichlicher Überlegung wurde ein neuer Befehl etabliert anstelle es in das normale graph-Environment einzubetten. Er funktioniert mit der Syntax:

env: wgraph

```

1 \begin{wgraph}{1}[]{}[0pt][\caption{Wichtiger Graph}]
2 \plotline{\x*\x}
3 \end{wgraph}
```

Die Signatur hierbei lautet: {ORIENTATION} [Argumente] [tikz-Argumente] [width] [wrapfig-zusatz].

2.3 3D-Plots WAR Ausstehend

Ich bin freier Platz :D

GRAFIKEN

ETLICHE VEREINFACHUNGEN UND ANDERE FREUDEN :D

VER 1.0.2

Alle folgenden Pfade sind relativ zu Data/Graphics/...

3.1 Grundlegende Symboliken

All diese Pakete werden über Tikz-Core/_LILLY_TIKZ_SYMBOLS eingebunden.

Dieses Paket liefert grundlegende, mal mehr und mal weniger, nützliche Tikz-Grafiken, welche zum Großteil aus denen in der Vorlesung verwendeten Grafiken entstanden sind. Alle diese Grafiken benötigen TikZ.

Allein deswegen ist es von Relevanz, die bessere Integration von TikZ - externalize und allgemein der Möglichkeit zur deaktivierung von TikZ voranzutreiben. Hierzu wird das ganze hier auch nochmals als **TODO** markiert und es wird darum gebeten, sich damit zu beschäftigen ©

3.1.1 Die Ampeln

Diese Definitionen befinden sich in der Datei: Tikz-Core/_LILLY_TIKZ_AMPEN

An sich handelt es sich hierbei um ein kleines Shortcut-Sammelsurium für Ampeln:

\ampelG

\ampelY

\ampelR

\ampelH

- ◇ \ampelG (●)
- ◇ \ampelY (●)
- ◇ \ampelR (●)
- ◇ \ampelH (○)

Explizit verwendet werden sie in zum Beispiel in den Erklärungen zum Moore-&Mealy-Automaten auf Basis der Ampelschaltung (●●○).

3.1.2 Emoticons WAR Ausstehend

Dieses Paket soll weitere lustige Begleiter im Textgeschehen zur Verfügung stellen:



3.1.3 Utility WAR Ausstehend

Dieses Paket soll die bisher von FontAwesome verwendeten Symbolen ersetzen und durch eigens erstellte Grafiken ersetzen.

3.1.4 Titleimages WAR Ausstehend

Dieses Paket soll die ganzen Titelgrafiken der Mitschriften enthalten und zur Verfügung stellen. Es soll mehr als Testpaket verstanden werden, Skripte werden dennoch die Grafik aus einer bereits generierten PDF beziehen.

3.2 Diagramme & Graphen

3.2.1 Graphen







Diese Definitionen befinden sich in der Datei: Tikz-Core/_LILLY_TIKZ_GRAPHEN

Auch wenn bereits TikZ hierfür Optionen anbietet, wurde aus optischen Gründen dieses Paket hinzugefügt! Grundlegend wird für den Radius aller Polygone empfohlen `\POLYRAD` zu verwenden (Standardmäßig: 1.61cm). Weiter definiert diese Bibliothek etliche sogenannte `graphdots`, welche alle nur in einer `tikzpicture`-Umgebung funktionieren (TODO: change - new environment), allen voran der normale `\graphdot(5)`. Er nimmt 5 Argumente entgegen: Füll-Farbe, Position, Text, Node-Name, Rand-Farbe. Analog hierzu definiert sich der `\tgraphdot(5)`-Befehl, welcher im Unterschied hierzu das erste Argument zwar entgegennimmt, aber ignoriert, da er eine `fill opacity` von 0 besitzt (TODO: selber Befehl mit optionalem Argument `fill opacity`).

Alle weiteren `graphdots` sind nun nichts weiteres als Shortcuts für die eben genannten Befehle:

◇ `\oragraphdot` (¹)

◇ `\blugraphdot` (²)

Analog hierzu: `\gregraphdot` () , `\purgraphdot` () , `\golgraphdot` () , `\blagraphdot` () , `\nographdot` () , `\margraphdot` () .

Ist zudem `\LILLYxMODExEXTRA` auf `\true` gesetzt, so wird `\graphPOI(8)` so konfiguriert, dass er die zugehörige Grafik anzeigt, ist dies nicht der Fall (in anderen Worten: `\LILLYxMODExEXTRA=\false`), so wird das Bild sowie zugehöriger Rahmen und Hyperlink nicht eingebracht. Grundlegend wurde dieser Befehl speziell für das Erstellen von Zeitleisten eingeführt und funktioniert nach folgendem Schemata (hier explizit mit Grafik):

```
1 \begin{tikzpicture}[scale=0.75,
2   every node/.style={transform shape}]
3   \graphPOI{(0,0)}{purple}{1999 n.Chr.}
4     {Florian Sihler}
5     {Florian Sihler ist der Autor dieses Dokuments}
6     {Data/2003.jpg}
7     {https://github.com/EagleoutIce/Quickblit}
8     {Deutschland};
9 \end{tikzpicture}
```

— 1999 n.Chr. Deutschland
Florian Sihler: Florian Sihler ist der Autor
 dieses Dokuments



Hier wurde aus Platzgründen die Größe angepasst.

¹`\tikz{\oragraphdot{(0,0)}{42}{a}};` – Zum E. d. Textzeile v. um -0.35baselineskip verschoben.

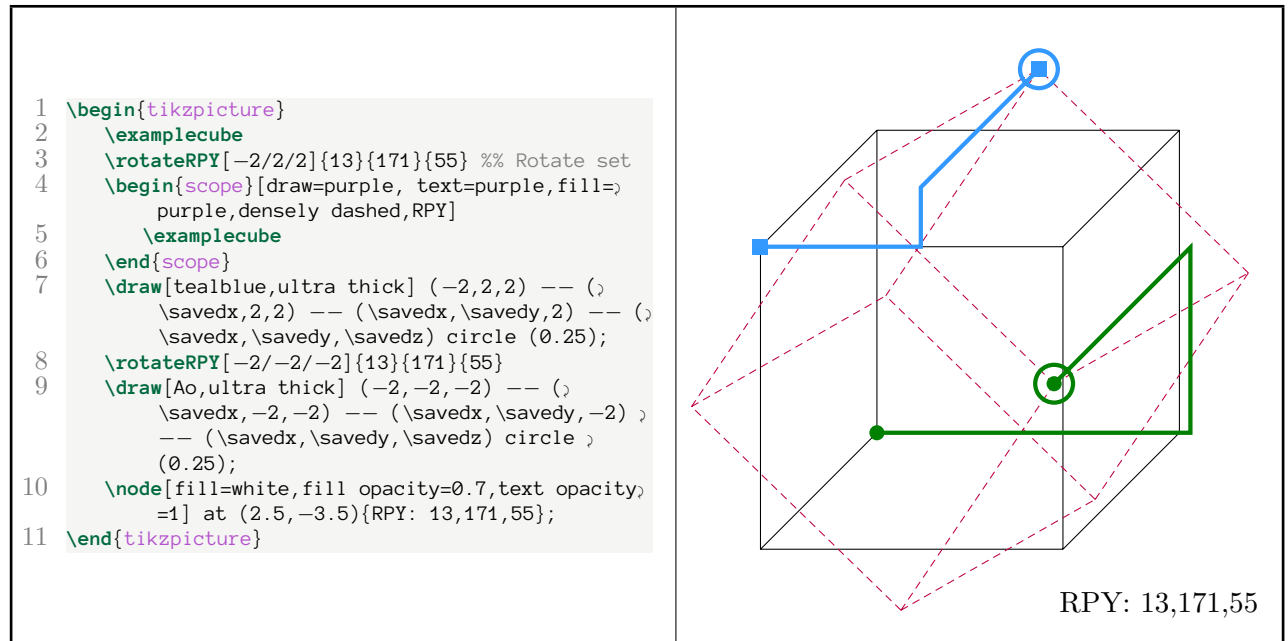
²`\tikz{\blugraphdot{(0,0)}{42}{a}};` – Zum E. d. Textzeile v. um -0.35baselineskip verschoben.

3.2.2 Rotation

Diese Definitionen befinden sich in der Datei: Tikz-Core/_LILLY_TIKZ_ROTATION

`\rotateRPY`

Diese Datei liefert nur den Befehl `\rotateRPY[1](3)` (roll, pitch, yaw) und den TikZ-Style `RPY`. Diese werden verwendet um erstellte TikZ Grafiken zu drehen und dementsprechend anzupassen. Dieser Code entstammt der Feder von David Carlisle und Tom Bombadil³ und wird hier beispielhaft illustriert:



³<https://tex.stackexchange.com/questions/67573/tikz-shift-and-rotate-in-3d>

3.2.3 Automaten WAR Work in Progress

Diese Definitionen befinden sich in der Datei: Tikz-Core/_LILLY_TIKZ_AUTOMATEN

Obwohl bereits TikZ eine Bibliothek für das Generieren von Automaten zur Verfügung stellt, wurde dieses (Work in Progress) Paket erstellt um darauf aufbauend schnell Automaten erstellen zu können. Der Grundbefehl `\loopTo[1](4)` erhält die Argumente: `[looseness]{Orientierungswinkel:360}{node-name}{Text}{Orientierung}`. Die weiteren 4 Befehle vereinfachen nun die Nutzung dieses Befehls für die häufigsten Fälle:

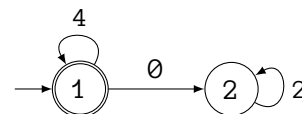
<code>\loopTop</code>	
<code>\loopRight</code>	◇ <code>\loopTop[1](2)</code> - Schleife oben
<code>\loopLeft</code>	◇ <code>\loopRight[1](2)</code> - Schleife rechts
<code>\loopBot</code>	◇ <code>\loopLeft[1](2)</code> - Schleife links
	◇ <code>\loopBot[1](2)</code> - Schleife unten

Für sie alle gelten nur noch die Argumente: `[looseness]{node-name}{Text}`. Im folgenden sei eine beispielhafte Verwendung gezeigt (der Automat muss keinen Sinn ergeben es soll lediglich die Nutzung verdeutlicht werden):

```

1 \begin{tikzpicture}[scale=1,
2   every node/.style={minimum size=12pt,transform shape},
3   state/.style={circle, draw, minimum size=20pt},
4   every path/.style={draw, -latex},
5   every initial by arrow/.style={-latex, initial text=}]
6
7   \node[initial,accepting,state] (1) at (180:1){\T{1}};
8   \node[state] (2) at (0:1){\T{2}};
9
10  \draw (1) to node[pos=0.5,above,sloped]{\T{0}} (2);
11  \loopTop[4]{1}{\T{4}};
12  \loopRight[4]{2}{\T{2}};
13 \end{tikzpicture}

```



Natürlich soll dieses Erstellen noch weiter stark vereinfacht werden. Des Weiteren wird darüber nachgedacht, einen akzeptierten Endzustand klarer zu markieren (Linien dicker, mehr abstand etc). Der Traum wäre, dass das Erstellen eines Automaten wie folgt funktioniert:

```

1 \begin{Automat}
2   \STATE[1]{180:1}{1};
3   \state[2]{0:1}{2};
4
5   \draw (1) to node[midway,above]{0} (2);
6
7   \loopTop[4]{1}{\T{4}};
8   \loopRight[4]{2}{\T{2}};
9 \end{Automat}

```

Die Befehle `\[1]{2}STATE` und `\[1]{2}state` sollen hierbei automatisch hochzählen können - pro Automat - aber über das optionale Argument lesbar einer Zahl zugewiesen werden. Die Umgebung `Automat` soll hierbei zusätzlich auch handhaben, dass automatisch alle Nodes mithilfe von `\T` geschrieben werden. Der entstehende Automat soll optisch identisch zum obigen sein.

3.2.4 Schaltkreise WAR Ausstehend

3.2.5 Neuronen WAR Work in Progress

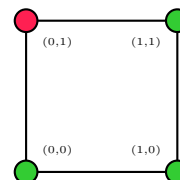
Diese Definitionen befinden sich in der Datei: Tikz-Core/_LILLY_TIKZ_NEURONS

Da vor allem mit *Formale Grundlagen* der Wunsch danach aufkam, neuronale Netze schnell zu Texen, wurde dieses Paket entwickelt um das Paket mit den Schaltkreisen so zu erweitern, dass es erlaubt Perzeptronen darin einzubauen, das Paket an sich befindet sich ebenfalls im Work in Progress-Status.

Es wurde bisher auch nur durch das Bereitstellen eines einzelnen Befehls implementiert: `\neuronSquare(4)`. Dieser funktioniert seinerseits lediglich in einer *tikzpicture*-Umgebung und zeichnet nichtmal ein Neuron, sondern lediglich die 2-D Repräsentation eines booleschen Raums, der wiedergibt unter welchen Eingabevektoren das Perzeptron welchen Wert zurückliefert. Die 4 Parameter, die hierzu `\neuronSquare` benötigt, entsprechen der jeweiligen Binärdarstellung der Eingabevektoren. Eine beispielhafte Anwendung ist hier zu finden:

`\neuronSquare`

```
1 \begin{tikzpicture}
2   \neuronSquare[limegreen]{Awesome}{limegreen}{limegreen};
3 \end{tikzpicture}
```



Das Schaltkreise-Paket ist ebenfalls noch nicht in LILLY integriert. Es befindet sich ebenfalls in einem Anfangsstadium und deswegen wird auch hierbei um Mithilfe bei der Weiterentwicklung gebeten.

Geplant ist es, die Darstellung der Informationen so zu vereinfachen, dass es (für alle Umgebungen) mithilfe von `tikzset` genügt zu schreiben: `\neuronSquare{T}{F}{T}{T}`. Zudem sollte der Namen des Befehls abgeändert werden und auch für 1-, 3- und 4-dimensionale Räume eine Option anbieten (siehe für 4D: Titelgrafik *Grundlagen der Rechnerarchitektur*), die dann über einen einfacheren Namen abgegriffen werden kann. Weiter sollen dann *Formale Grundlagen* und *Grundlagen der Rechnerarchitektur* (boolesche Räume) diese Befehle nutzen anstelle der dafür eigens implementierten Grafiken. Weiter soll es über ein optionales Argument möglich sein die Position relativ zu bestimmen!

4

FARBEN

VIELE VIELE BUNTE FARBEN

VER 1.0.4

Alle folgenden Pfade sind relativ zu `Data/...`

Im folgenden wird beschrieben wie grundlegend die Einbettung eines neuen Farbprofils ab VER 1.0.4 funktioniert. Bitte beachte, dass vor dieser Version ein Farbprofil noch alle Farben überschreiben und liefern musste, während seit dieser Version mit dem Überschreiben der Standard-Farben gearbeitet wird. Wichtig ist:

Jedes Farbprofil kann eigene Farben hinzufügen - hiervon wird aber stark abgeraten, da somit nichtmehr die Design-Unabhängigkeit von LILLY garantiert ist!

4.1 Die normalen Farbprofile

4.1.1 Das Standardfarbprofil

Diese Definitionen befinden sich in der Datei: `Colors/_LILLY_DEFAULT_COLORPROFILE`

\LILLYxColorxInject Dieses Farbprofil wird nur geladen, wenn die Variable \LILLYxColorxInject nicht definiert ist. Es selbst bindet die Pakete `xcolor` und `pgf` (?) mit ein und definiert seinerseits eine Menge Farben, die zum einen einfach gut aussehen ☺ und zum anderen von Einzelpersonen gewünscht wurden. Alle Farben, welche LILLY darüber zur Verfügung gestellt bekommt, werden im folgenden aufgelistet:

● Aureolin ($r: 253, g: 238, b: 0$)	● Azure ($r: 0, g: 127, b: 255$)
● Amber ($r: 255, g: 191, b: 0$)	● bondiBlue ($r: 0, g: 149, b: 182$)
● ChromeYellow ($r: 255, g: 167, b: 0$)	● antiVeg ($r: 190, g: 238, b: 239$)
● Coquelicot ($r: 255, g: 56, b: 0$)	● DarkOrchid ($r: 104, g: 34, b: 139$)
● Cinnabar ($r: 227, g: 66, b: 52$)	● Veronica ($r: 180, g: 82, b: 205$)
● BrightMaroon ($r: 195, g: 33, b: 72$)	● Amethyst ($r: 153, g: 102, b: 204$)
● Cherry ($r: 222, g: 49, b: 99$)	● AntiqueFuchsia ($r: 145, g: 92, b: 131$)
● AlizarinCrimson ($r: 227, g: 28, b: 54$)	● BritishRacingGreen ($r: 0, g: 66, b: 37$)
● Amaranth ($r: 229, g: 43, b: 80$)	● DatmouthGreen ($r: 0, g: 105, b: 62$)
● AmericanRose ($r: 255, g: 3, b: 62$)	● Ao ($r: 0, g: 128, b: 0$)
● Awesome ($r: 255, g: 32, b: 82$)	● AppleGreen ($r: 141, g: 82, b: 0$)
● BrightPink ($r: 255, g: 0, b: 127$)	● BrightGreen ($r: 102, g: 255, b: 0$)
● DebianRed ($r: 215, g: 10, b: 83$)	● LightGray ($r: 224, g: 224, b: 224$)
● Crimson ($r: 220, g: 20, b: 60$)	● AuroMetalSaurus ($r: 110, g: 127, b: 128$)
● DarkMidnightBlue ($r: 0, g: 51, b: 102$)	● Charcoal ($r: 54, g: 69, b: 79$)

Bemerkung 4.1 – Kompatibilität

Weiter gibt es die folgenden Farben, welche aus Kompatibilitätsgründen aus dem `eagleStudiPackage` übernommen wurden:

● dpurple (<i>r: 86, g: 60, b: 92</i>)	● gold (<i>r: 255, g: 215, b: 50</i>)
● ddpurple (<i>r: 128, g: 0, b: 128</i>)	● dgold (<i>r: 235, g: 198, b: 13</i>)
● beauty (<i>r: 104, g: 55, b: 107</i>)	● limegreen (<i>r: 51, g: 204, b: 51</i>)
● candypink (<i>r: 227, g: 112, b: 122</i>)	● skyblue (<i>r: 60, g: 179, b: 113</i>)
● thered (<i>r: 255, g: 47, b: 47</i>)	● tealblue (<i>r: 51, g: 153, b: 255</i>)
● dorange (<i>r: 255, g: 102, b: 0</i>)	○ superlightgray (<i>r: 240, g: 240, b: 240</i>)
● mint (<i>r: 255, g: 128, b: 0</i>)	

Sie sollten nicht mehr verwendet werden!

Weiter definiert dieses Farbprofil die Farben, welche LILLY für Links, Boxen usw. verwenden soll. Alle diese Befehle sollten auch bei eigenen Implementationen und Erweiterungen angewendet werden, darum folgt hier eine Auflistung für die Boxen (alle Befehle beginnen mit `\LILLYx`):

● ColorxDefinition (<i>DebianRed</i>)	● ColorxUebungsaufgabe (<i>Veronica</i>)
● ColorxSatz (<i>Ao</i>)	● ColorxZusatzuebung (<i>Veronica</i>)
● ColorxBeweis (<i>DarkMidnightBlue</i>)	● LINKSxMainColor (<i>DebianRed!85!black</i>)
● ColorxLemma (<i>DarkMidnightBlue</i>)	● LINKSxCiteColor (<i>DarkMidnightBlue</i>)
● ColorxBemerkung (<i>Charcoal</i>)	● LINKSxUrlColor (<i>DarkMidnightBlue</i>)
● ColorxZusammenfassung (<i>ChromeYellow</i>)	● TITLExCOLOR (<i>DebianRed</i>)
● ColorxBeispiel (<i>Aureolin</i>)	

Weiter gibt es noch die Farbe: `\LILLYxLINKSxMainColorDarker` (●). Sie wird gemäß: `\LILLYxLINKSxMainColor!90!black` generiert.

Beispielhaft lässt sich die Definitionsfarbe mit: `\LILLYxColorxDefinition` abfragen (●)

4.1.2 Das Druckprofil

Diese Definitionen befinden sich in der Datei: `Colors/_LILLY_PRINT_COLORPROFILE`

Auch dieses Profil definiert seine Farben nur, wenn `\LILLYxColorxInject` nicht definiert ist! Aus Kompatibilitätsgründen zu Versionen vor VER 1.0.4 definiert dieses Profil grundlegend die gleichen Farben wie das Standardfarbprofil. Hier wird nur auf die alternativen LILLY-Farben eingegangen, da sich die Profile nur hierin unterscheiden:

● ColorxDefinition (<i>DebianRed</i>)	● ColorxBemerkung (<i>Charcoal</i>)
● ColorxSatz (<i>Charcoal</i>)	● ColorxZusammenfassung (<i>ChromeYellow</i>)
● ColorxBeweis (<i>Charcoal</i>)	● ColorxBeispiel (<i>Charcoal</i>)
● ColorxLemma (<i>Charcoal</i>)	● ColorxUebungsaufgabe (<i>Charcoal</i>)

- ColorxZusatzuebung (Charcoal)
- LINKSxMainColor (Charcoal)
- LINKSxCiteColor (Charcoal)
- LINKSxUrlColor (Charcoal)
- TITLExCOLOR (DebianRed)

Die Farbe \LILLYxLINKSxMainColorDarker (●) wird hier mithilfe von: \LILLYxLINKSxMainColor!95!bla generiert.

4.2 Weitere Planungen

- ◇ Elysium WAR Ausstehend
- ◇ Besseres Druckprofil WAR Ausstehend
- ◇ Weitere Farben WAR Ausstehend - Generische Farben wie „Rot“ auch als Befehl - zudem Lösung für Druckversion, sodass nirgendwo steht - der „Rote Kreis“ - wenn er dann eigentlich schwarz ist.

LISTINGS

IST THIS... THE MATRIX?

VER 1.0.0

Sei es nun *Formale Grundlagen*, *Einführung in die Informatik* oder *Grundlagen der Rechnerarchitektur*, in jeder Vorlesungsreihe war es von Relevanz Quelltexte mit Syntax-Highlighting zu versehen. Hierfür verwendet LILLY die Bibliothek `listings` und fügt einige Styles und ein paar Sprachen hinzu, die ebenfalls frei gewählt werden können. Aktuell ist die Implementation an vielen Stellen noch weit weg von perfekt. So ist es in GDRA zum Beispiel immer noch vonnöten das Highlighting, von zum Beispiel `addiu`, mithilfe von `*\mipsADD*` einzubinden. An einer Lösung hierfür wird aktuell gearbeitet, siehe weiter unten.

5.1 Die grundlegenden Eigenschaften

5.1.1 Grundlegendes Design

Diese Definitionen befinden sich in der Datei: `Listings/_LILLY_LISTINGS_STYLE`

LILLY verwendet nicht das normale `listings`-Paket, sondern greift auf das erweiterte Paket `listingsutf8` zu. Weiter definiert es für alle klassischen Umlaute die nötigen Befehle mithilfe von `\lstset{literature=...}`. Dieser Code wurde folgender Seite entnommen: https://en.wikibooks.org/wiki/LaTeX/Source_Code_Listings.

Um dynamisch zu bleiben bindet LILLY nicht einfach verschiedene Stile ein, sondern Dateien, welche dann für sich definieren, welche Stile und Sprachen zusätzlich zur Verfügung stehen.

Mithilfe von `\LILLYxListingsxLang` kann man das jeweilige Paket auswählen. Dieses Paket wird über den klassischen `\input{}`-Befehl eingebunden und zwar über folgende Anweisung:

```
1 \input{\LILLYxPATHxLISTINGS/Languages/_LILLY_LANG_\LILLYxListingsxLang}
```

Standardmäßig wird so das MAIN-Paket geladen, welches ebenfalls die LANG-Signatur besitzt - dies soll geändert werden.

Bisher ist das Einbinden neuer Stile noch recht starr, da der Dateipfad bis auf das Suffix vorgegeben ist. Dies sollte geändert werden

5.1.2 Das MAIN-Paket

Diese Definitionen befinden sich in der Datei: Listings/Languages/_LILLY_LANG_MAIN

showstringspaces	true
basicstyle	\LILLYx1stTypeWriter
numbers	left
escapeinside	{!*}{*}}
frame	single
language	assembler
numberstyle	\small\color{gray}
keywordstyle	\color{purple}\bfseries
commentstyle	\color{gray}
stringstyle	\color{mint}
extendedchars	true


Die allgemeine TypeWriter-Schriftart wird mithilfe von \LILLYx1stTypeWriter auf AnonymousPro gesetzt. Sie wird auch hier für die Dokumentation verwendet. Weiter lädt es folgende Sprachen & Stile (die Differenzierung ist hierbei noch nicht abgeschlossen):

- ◇ assembler
- ◇ pseudo
- ◇ pseudo_ - nospace
- ◇ bash
- ◇ latex

Zudem lädt MAIN noch das Paket MIPS, auf welches weiter unten noch weiter eingegangen wird. Weiter wird \1st@PlaceNumber modifiziert und es werden einige grundlegende Einstellungen getätigt, welche sich in der linken Tabelle wiederfinden lassen. Im Folgenden werden die einzelnen hierrüber eingebundenen Sprachen nicht weiter beschrieben - hierzu gibt es eigene Sektionen weiter unten...

5.1.3 Das MIPS-Paket

Diese Definitionen befinden sich in der Datei: Listings/Languages/_LILLY_LANG_MIPS

Dieses Paket wurde vor allem im Rahmen von *Grundlagen der Rechnerarchitektur* erstellt und bindet das Paket `caption` mit ein, um die Positionierung von Titeln zu vereinfachen. Weiter definiert es die Befehle \gitRAW und \git (Diese sollten besser verschoben werden TODO:). Sie fügen mithilfe von FontAwesome ein Github Symbol ein, welches auf ein Github-Repository verweist, indem sich alle in *Grundlagen der Rechnerarchitektur* verwendeten Codes befinden (https://www.github.com/EagleoutIce/MIPS_UniUlm_Examples/: ). Dieses Paket wird vom MAIN-Paket eingebunden und definiert weiter folgende Stile:

MIPS

Syntax-Highlighting für alle grundlegende MIPS-Befehle - verwendet 6 verschiedene Farben für verschiedene Arten von Keywords:

- Zeichenketten (*candypink*)
- Befehle (*purple*)
- Register (*tealblue*)
- Direktiven (*dgold*)
- Spezielle Befehle (*limegreen*)
- Buzzwords (*thered*)
- Daten-Direktiven (*tealblue!60!black*)

Weiter setzt es die Position der Zeilennummern auf die rechte Seite.

MIPSSNIP

Funktioniert analog zu MIPS, aber definiert das Design für kurze Ausschnitte.

5.2 Die mitgelieferten Erweiterungen

5.2.1 assembler

Definitionen aus der Datei: Listings/Languages/_LILLY_LANG_assembler

showstringspaces	true
basicstyle	\LILLYx1stTypeWriter
keywordstyle	\color{purple}\bfseries
commentstyle	\color{gray}
stringstyle	\color{mint}
extendedchars	true
comment	[l]{//}
morecomment	[s]{/*}{*/}
morestring	[b]'

mit folgendem Style:

\color{tealblue!80!black}\bfseries.

Diese Sprache liefert eine seltsame Mischung an Assembler-Befehlen, die in *Grundlagen der Rechnerarchitektur* zum Teil als Pseudo-Assembler-Befehlssatz verwendet wurden. Die definierten Schlüsselwörter lauten:

while, if, r, ld, st, sr, sl, beq, bnq, add, sub,

Weiter werden ndkeywords definiert nop, X, acc,

5.2.2 pseudo

Definitionen aus der Datei: Listings/Languages/_LILLY_LANG_pseudo

showstringspaces	false
basicstyle	\LILLYx1stTypeWriter
keywordstyle	\color{purple}\bfseries
commentstyle	\color{gray}
stringstyle	\color{mint}
extendedchars	true
comment	[l]{//}
morecomment	[s]{/*}{*/}
morestring	[b]'

Diese Sprache liefert die Befehle für die Pseudo-Programmiersprache in *Formale Grundlagen*. Die definierten Schlüsselwörter lauten:

INPUT, REPEAT, ELSE, UNTIL, OR, END, FOR, IF, END

5.2.3 pseudo_nospace WAR Veraltet

Definitionen aus der Datei: Listings/Languages/_LILLY_LANG_pseudo_nospace

Definiert bis auf die Option sensitive die selben Dinge wie pseudo und ist nur noch aus Kompatibilitätsgründen vorhanden.

5.2.4 bash

Definitionen aus der Datei: Listings/Languages/_LILLY_LANG_bash

Dieser Stil wurde für die Dokumentation erstellt und wird mit ihr erweitert und verfeinert. Es ist davon auszugehen, dass alle in dieser Dokumentation verwendeten Befehle zur Verfügung stehen. Bei Bedarf kann dieses Paket gerne erweitert und ausgebaut werden. Bisher definiert es Befehle wie: mkdir, texhash, make, apt, Parameter wie -p, -dir, print, install und Dokumentbezeichnern wie: lilly_compile.sh, sudo.

FunFact: Für Variablen mit \$ wird die Farbe *antiVeg* verwendet (● - #BEEEEF)

5.2.5 latex

Definitionen aus der Datei: Listings/Languages/_LILLY_LANG_latex

Dieser Stil wurde ebenfalls für die Dokumentation erstellt und wird genauso erweitert und verfeinert. Deswegen wird in diesem Rahmen hier ebenfalls keine vollständige Auflistung stattfinden. Lediglich eine Auflistung der verwendeten Stile für die einzelnen Schlüsselwörter:

- | | |
|---------------------------------------|------------------------------|
| ● Zeichenketten (<i>Amber</i>) | ● Parameter (<i>Amber</i>) |
| ● Befehle (<i>black</i> – \bfseries) | ● Umgebungen (<i>Ao</i>) |
| ● Lilly (<i>Awesome</i> – \bfseries) | ● Kommentare (<i>gray</i>) |

Es werden keine Zeilennummern angezeigt.

6

BOXEN

BOXES IN BOXES IN BOXES IN BOXES...

VER 1.0.0

6.1 Grundlegendes

6.1.1 Eine kleine Einführung

Die 3 Standard-Designs, welche mit LILLY ausgeliefert werden lauten wie folgt:

DEFAULT	ALTERNATE	LIMERENCE
<div>Satz 6.1 Nice</div> <div>Superwichtig</div>	<div>Satz 6.2 – Nice</div> <div>Superwichtig</div>	<div>Satz 6.3 – Nice</div> <div>Superwichtig</div>

Auch wenn sie hier explizit forciert wurden ist es grundlegend möglich (und auch so gedacht) sie mithilfe des Makefiles konfigurieren. Die allgemeine Syntax hierfür lautet:

```
1 make "BOXMODE=<Name>"
```

wobei <Name> mit einem der oben stehenden Bezeichner ersetzt wird. Die Bezeichner werden vom weiter unten näher beschriebenen Box-Controller wie folgt aufgelöst:

```
1 \input{\LILLYxPATHxDATA/POIs/_LILLY_BOXES_\LILLYxBOXxMODE}
```

Über genau dieses Verfahren lassen sich auch beliebig die Box-Designs erweitern.

LILLY lädt
übrigens nicht
DEFAULT
sondern immer
DEFAULT(init)!

Natürlich wäre es
schöner auch an-
dere Verzeichnisse
zuzulassen und
hierbei dann den
gesamten Pfad
anzugeben - dies
ist aber bisher
auch TODO:

6.1.2 Der Box-Controller

Diese Definitionen befinden sich in der Datei: `Controllers/_LILLY_BOX_CONTROLLER`

`\LILLYxBOXxMODE`

Alle Box-Desings werden über den Box-Controller geladen, der über `\LILLYxBOXxMODE` die Möglichkeit zur Verfügung stellt die jeweilige POI-Datei zu laden (TODO: LINK). Er definiert ein gigantisches Paket an Befehlen (TODO: pgf foreach) die allerdings für jeden Boxtyp identisch sind. allgemein werden: `\LILLYxBOXx*xLock` und `\LILLYxBOXx*xEnable` für alle Boxen definiert. So kann man zum Beispiel durch das Setzen von `\LILLYxBOXxDefinitionxEnable` auf FALSE das Anzeigen von Definitionsboxen deaktivieren (Information: Sie werden einfach entfernt, es wird kein adäquater Platzhalter als Ersatz eingefügt) und durch das Setzen von `\LILLYxBOXxBeispielxLock` aufsection das Nummerieren der Box auf die Sektionen festlegen (TRUE für ungebunden). Weiter definiert es die folgenden Box-Environments:

Definition 3 – Titel

moin

```
1 \begin{definition}[Titel]
2     moin
```

```
3 \end{definition}
```

Definition 4 – Titel

mein

```
1 \begin{definition*}[Titel]
2     mein
3 \end{definition*}
```

Bemerkung 6.1 – Titel

mein

```
1 \begin{bemerkung}[Titel]
2     mein
3 \end{bemerkung}
```

Beispiel 6.1 – Titel

mein

```
1 \begin{beispiel}[Titel]
2     mein
3 \end{beispiel}
```

Satz 6.4 – Titel

mein

```
1 \begin{satz}[Titel]
2     mein
3 \end{satz}
```

Beweis 6.1 – Titel

mein

```
1 \begin{beweis}[Titel]
2     mein
3 \end{beweis}
```

Lemma 6.1 – Titel

mein

```
1 \begin{lemma}[Titel]
2     mein
3 \end{lemma}
```

Zusammenfassung 6.1 – Titel

mein

```
1 \begin{zusammenfassung}[Titel]
2     mein
3 \end{zusammenfassung}
```

Aufgabe 3 – Titel

mein

```
1 \begin{aufgabe}[Titel][3]
2     mein
3 \end{aufgabe}
```

Nicht richtig darstellbar aber weiter existiert:

```
1 \begin{uebungsblatt}[Titel][2]
2     mein
3 \end{uebungsblatt}
```

\LILLYxBOXx*xBox

Für bisher leider noch nicht alle Boxen wird zudem der Befehl: `\LILLYxBOXx*xBox` definiert. Bisher unterstützt werden:

- ◇ Bemerkung
- ◇ Beweis
- ◇ Uebungsblatt
- ◇ Beispiel
- ◇ Aufgabe

Setzt man den Wert auf `FALSE` so wird das sogenannte `plain-Design` angewendet, welches jedes Design wieder selbst definieren kann! (TODO: custom Box counters).

Zudem existieren aus Kompatibilitätsgründen auch noch die alten Befehle aus dem `eagleStudiPackage`:

```

\DEF \DEF(2), \BEM(2), task,...
\BEM
env: task
\inputUB
\inputUBS

```

Mit **VER 1.0.3** wurden in LILLY zudem Kurzbefehle für das Einbinden von Übungsblättern integriert: `\inputUB(3)` (mit Signatur: `{Name}{Nummer}{Pfad}`) und `\inputUBS(3)` (analog für `uebungsblatt*`)

6.2 Die Boxmodi

6.2.1 Default-Design

Mit **VER 1.0.0** stellt dieses Design den Urvater dar. Im Folgenden wird auf die genaue Implementation eingegangen:

Auf Basis des Pakets `tcolorbox` definiert LILLY das Design `LillyxBOXxDesignxDefault` - auf das Großschreiben von Lilly wurde hier bewusst verzichtet - mit folgender Implementation:

```

1 \tcbset{LillyxBOXxDesignxDefault/.style={enhanced jigsaw, pad before >
   break*=2mm %
2   pad after break=2mm, lines before break=4, before skip=0pt, >
   boxrule = 0mm, toprule=0.5mm,%
3   bottomtitle=0.5mm,bottomrule=1.2mm, after skip=0pt, enlarge top by>
   =\baselineskip,%
4   enlarge bottom by=\baselineskip, sharp corners=south, enforce >
   breakable}%
5 }

```

Bisher definiert LILLY die Counter über die Einstellung `auto counter` - dies soll aber bald auf das vom `eagleStudiPackage` Package verwendete `counter`-Verfahren umgestellt werden. Bis dato sieht eine exemplarische Definition einer Box wie folgt aus:

```

1 \DeclareTColorBox[auto counter]%
2   {LILLYxBOXxDefinition}%
3   { 0{ } 0{Definition \thetcbcounter~} 0{drop fuzzy shadow} }%
4   {LillyxBOXxDesignxDefault, colback=\LILLYxColorxDefinition!5!>
   white,%
5   colframe=\LILLYxColorxDefinition, #3,%
6   title={%
7     \begin{minipage}[t][\baselineskip][l]{\textwidth}%
8       \textbf{\textsc{#2}} \hfill {\textbf{#1}}%
9     \end{minipage}%
10  }%
11 }

```

Hiervon weichen nur 2 Definitionen ab. Die der Aufgaben-Box:

```

1 \DeclareTColorBox{LILLYxBOXxAufgabe}{0{ } 0{ } 0{ }}{enforce breakable,%
2   colback=white,colframe=black!50,boxrule=0.2mm,%
3   attach boxed title to top left={xshift=1cm,yshift*=1>
   mm-\tcboxedtitleheight},%
4   varwidth boxed title*=-3cm,%
5   boxed title style={

```

```

6      frame code={
7          \path[fill=white!30!black]%
8              ([yshift=-1mm,xshift=-1mm]frame.north west)%
9              arc[start angle=0,end angle=180,radius=1mm]%
10             ([yshift=-1mm,xshift=1mm]frame.north east)%
11             arc[start angle=180,end angle=0,radius=1mm];
12      \path[left color=white!40!black,right color=white!40!black,
13            middle color=white!55!black]
14            ([xshift=-2mm]frame.north west) -- ([xshift=2mm]frame.north east)%
15            [rounded corners=1mm]-- ([xshift=1mm,yshift=-1mm]frame.north east)%
16            -- (frame.south east) -- (frame.south west)%
17            -- ([xshift=-1mm,yshift=-1mm]frame.north west)%
18            [sharp corners]-- cycle;%
19      },interior engine=empty,%
20  },
21  enhanced jigsaw, before skip=2mm,after skip=2mm,%
22  fonttitle=\bfseries, #3,%
23  title={#2 \ifthenelse{\equal{#1}{}}{\{--~\}#1}, %Aufgabe
24 }

```

Und die der Plain-Box:

```

1  \DeclareTColorBox{LILLYxBOXxAufgabexPlain}{O{} O{} O{}} {%
2      enforce breakable, enhanced jigsaw, before skip=2mm,after skip=2mm,%
3      colback=white,colframe=black!50,boxrule=0.2mm,fonttitle=\bfseries,%
4      #3,title={#2 \ifthenelse{\equal{#1}{}}{\{--~\}#1}%
5  }

```

7

JAKE

Jake! WOULD YOU GET ME THE CAKE PLEASE?...

VER 1.0.8

7.1 Grundlegendes

7.1.1 Entwicklung

Anfänglich wurde *Jake* als *installer* konzipiert, der einfach nur die mühselige Installation des Pakets abnehmen soll. Mittlerweile hat sich *Jake* allerdings weiterentwickelt und bietet das Potenzial für einiges mehr. Im Folgenden sei die Funktionsweise genauer erklärt. Zu beachten ist allerdings, dass *Jake* bisher nur für Linux und MacOS einen Installer und somit seine Funktionalität zur Verfügung stellt!

7.1.2 Die Installation

Jake zu installieren sollte normalerweise einem Kinderspiel gleichen. Notwendig sind hierfür auf allen bisher unterstützten Betriebssystemen (Debian-Basiertes Linux und MacOS) ein C++14 fähiger gcc-Compiler und make. Anschließend gilt es ins `jake_source`-Verzeichnis zu navigieren. Es befindet sich hier: `Lilly/Jake/jake_source`. In diesem Verzeichnis kann man nun `make` ausführen. Dies sorgt dafür, dass nicht nur `jake.cpp` zu einer ausführbaren Datei wird, sondern auch, dass `|lilly_jake|` systemweit zur Verfügung steht (sofern die verwendeten Konsole `bash`, `zsh` oder `iTerm` ist, bzw. im allgemeinen auf eine der folgenden Dateien zugreift: `.bashrc`, `.zshrc`, `.bash_profile`).

Damit gilt *Jake* als *installiert*.

7.1.3 Die Schnittstelle

Mit *Jake* interagiert es sich über die Konsole ganz einfach. Die Eingabe von `|lilly_jake|` zeigt eine Hilfe mit allen nötigen Optionen an. Nutzt man `bash` oder `zsh` so wird *Jake* übrigens bereits automatisch Vervollständigt.

Du hast die aktuelle Grenze der offnene Dokumentationswelt erreicht. „Still under construction“, wie man so schön sagt. Für aktuelle Informationen sollten aber die im Repository existierenden Readmes ausreichen.

8

AUSSICHT

DAS WUNDER DER SCHÖPF... EVOLUTION ☺

8.1 Bekannte Probleme

8.1.1 TikZ

Bisher ist es nicht möglich TikZ an sich zu deaktivieren (auch wenn es die Flagge dafür gibt). Weiter ist es nicht möglich die Option *externalize* zu aktivieren - Dies sollte unbedingt angegriffen werden!

8.2 Todos

8.2.1 Visuals

Es wäre schön (auch auf Basis von tcolorbox) einige Umgebungen zu haben, mit denen sich Grafiken oder Textabschnitte einfach positionieren lassen. So ist es lästig hierfür jedesmal minipages und unsicher hierfür jedesmal floatings zu verwenden.

8.2.2 Fehler

Das Paket sollte Befehle wie `\PackageInfo/Error/Warning` unterstützen und auch ausgeben - zudem sollte die komplette Dateistruktur robuster werden und auf Fehler reagieren können

8.2.3 Dateiaufteilung

Die Aufteilung von LILLY in verschiedene Dateien war zum Beibehalt der Übersicht unabdinglich, allerdings sollte diese Aufteilung einigen Kontrollblicken und Korrekturen unterzogen werden - zudem sollte in dem Rahmen das Implementieren neuer Designs/Codes vereinfacht werden - hierfür würde sich ein einfaches Skript anbieten, was neue Dateien (je nach Typ) automatisch an die richtige Stelle bringt. Weiter wäre es gut, wenn die Dateinamen nicht nur `.tex` o.ä. lauten würden

8.2.4 Road to CTAN

Es sollten die notwendigen Installationsdateien und Dokumentationen generiert und eingebracht werden - sodass Lilly automatisiert verwaltet werden kann.

8.2.5 Hoverover tooltips

Eine Idee war es bei Hyperlinks Kommentare mithilfe von Tooltips zu realisieren. Somit wäre es möglich auf den meisten Geräten schnell Informationen zu liefern mithilfe von: Ich bin ein toller Hyperlink.

8.2.6 Weitere

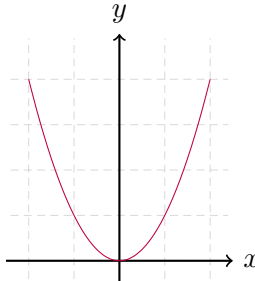
Siehe hier für weitere Todos: <https://github.com/EagleoutIce/LILLY/issues>

9.1.2 Spezifikation: Plots

Dieser Abschnitt beschreibt die Richtlinien, auf denen Plots in LILLY integriert werden sollen. Es wurden noch keine (TikZ) basierte Plot-Umgebungen in LILLY integriert.

graph-Environment:

Es soll ein graph-Environment existieren, was auf Basis von PGF das Erstellen folgender Grafiken immens vereinfachen soll:

Aktuell	Ergebnis	Wunsch
<pre> 1 \begin{tikzpicture}[scale=0.6] 2 \draw[help lines, color=gray!30, 3 densely dashed] (-2.4,-0.4) > 4 grid (2.4,4.9); 5 \draw[->,thick] (-2.5,0) -- (2.5,0) 6 node[right]{\$x\$}; 7 \draw[->,thick] (0,-0.5) -- (0,5) 8 node[above]{\$y\$}; 9 \draw[scale=1,domain=-2:2, 10 smooth,variable=\x,purple] 11 plot ({\x},{\x*\x}); \end{tikzpicture} </pre>		<pre> 1 \begin{graph}[scale= 2 =0.6,domain=-2:2] 3 \plotline[purple]{ 4 \x}{\x*\x}; 5 \end{graph} </pre>

Der Befehl `\plotline` soll hierbei nur in der Umgebung verfügbar sein (TODO: gleiches geplant mit PLA etc.).

Positionierung:

Für die Platzierung von Plots wurden 3 valide Positionen vorgesehen: Zentriert, Links (Text auf rechter Seite), Rechts (Text auf linker Seite). Diese Positionierungen können mithilfe von Floats realisiert werden, sollen aber auf jedenfall auch noch einen absoluten Modus zur Verfügung stellen (primär von zentriert analog zu `\[\]`). Zudem soll das `plot`-Environment selbstverständlich auch ohne Positionierung manuell eingebunden werden können!

9.2 Version VER 1.0.9

9.2.1 Installation in Linux

Für Versionen < 1.0.8 klicke hier: [klick mich!](#)

Da LILLY komplett auf einem Linux-Betriebssystem entwickelt wurde, gestaltet sich die Implementierung relativ einfach. Hierzu nutzen wir das Hilfsprogramm *Jake* welches selbst in C++ geschrieben wurde. Im Folgenden sind die Schritte kurz erklärt.

VER 1.0.8

Installation von *Jake* :

Eine ausführliche Erklärung von *Jake* selbst findest sich weiter hinten ([hier](#)) in dieser Dokumentation:

1. Navigiere mit dem Terminal in das Verzeichnis: `Lilly/Jake/jake_source`
2. Führe nun `make` aus um *Jake* zu kompilieren. Es wird vermutlich kurz dauern, aber danach wird dir das Programm `|lilly_jake|` zur Verfügung stehen.
3. Nun kannst du dein Terminal neu starten und von überall her `lilly_jake install` aufrufen. Dies sollte den Installationsprozess in Gang setzen.

Für ausführliche Informationen zur Installation konsultiere bitte die README-Datei in: `../Lilly/Jake/jake_source/README.md`.
Für Informationen zur Nutzung konsultiere: `../Lilly/Jake/README.md`

Sollte das Ganze fehlerfrei verlaufen sein, dann: Glückwunsch, du hast Lilly erfolgreich installiert! Betrachte im Falle eines Fehlers bitte erst die Readme-Dateien und die bereits beantworteten Fehler auf Github ([🔗](#)) bevor du einen neuen Fehler eröffnest oder mir eine Nachricht schreibst ☺.

Erstellen eines Makefiles:

Nun möchtest du natürlich auch ausprobieren ob die Installation funktioniert hat. Hierzu kannst du in das Testverzeichnis navigieren (`Lilly/Jake/tests`). Hier befinden sich eine Menge Dateien die in dieser Dokumentation auch als Beispiele benutzt werden. Du gibst nun folgendes in die Konsole ein:

```
1 lilly_jake test.tex
```

Jake erstellt nun ein entsprechendes Makefile für dich, welches du nun ausführen kannst:

```
1 make
```

Im Standardmäßig konfigurierten Ausgabe-Ordner `test-OUT` befindet sich nun eine entsprechende PDF Datei ☺.

Bemerkung 9.2 – make

Logischerweise muss damit auch `make` auf dem System vorhanden sein:

```
1 sudo apt install "make"
```

9.2.2 Installation in MacOS

Entspricht, dank *Jake* , der Linux-Installation.

Hierzu nutzen wir das Hilfsprogramm *Jake* welches selbst in C++ geschrieben wurde. Im Folgenden sind die Schritte kurz erklärt.

Installation von *Jake* :

Eine ausführliche Erklärung von *Jake* selbst findest sich weiter hinten (TODO: LINK) in dieser Dokumentation:

1. Navigiere mit dem Terminal in das Verzeichnis: `Lilly/Jake/jake_source`
2. Führe nun `make` aus um *Jake* zu kompilieren. Es wird vermutlich kurz dauern, aber danach wird dir das Programm `|lilly_jake|` zur Verfügung stehen.
3. Nun kannst du dein Terminal neu starten und von überall her `lilly_jake install` aufrufen. Dies sollte den Installationsprozess in Gang setzen.

Für ausführliche Informationen zur Installation konsultiere bitte die README-Datei in: `../Lilly/Jake/jake_source/README.md`. Für Informationen zur Nutzung konsultiere: `../Lilly/Jake/README.md`

Sollte das Ganze fehlerfrei verlaufen sein, dann: Glückwunsch, du hast Lilly erfolgreich installiert! Betrachte im Falle eines Fehlers bitte erst die Readme-Dateien und die bereits beantworteten Fehler auf Github ([🔗](#)) bevor du einen neuen Fehler eröffnest oder mir eine Nachricht schreibst ☺.

Erstellen eines Makefiles:

Nun möchtest du natürlich auch ausprobieren ob die Installation funktioniert hat. Hierzu kannst du in das Testverzeichnis navigieren (`Lilly/Jake/tests`). Hier befinden sich eine Menge Dateien die in dieser Dokumentation auch als Beispiele benutzt werden. Du gibst nun folgendes in die Konsole ein:

```
1 lilly_jake test.tex
```

Jake erstellt nun ein entsprechendes Makefile für dich, welches du nun ausführen kannst:

```
1 make
```

Im Standardmäßig konfigurierten Ausgabe-Ordner `test-OUT` befindet sich nun eine entsprechende PDF Datei ☺.

Bemerkung 9.3 – make

Logischerweise muss damit auch `make` auf dem System vorhanden sein:

```
1 sudo apt install "make"
```