

Lilly is a Latex Lovable Yogurt

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

## Dokumentation – Version 2.0.0

Autor & Instandhaltung:

**Florian Sihler** (florian.sihler@web.de)

21. August 2019

### *Abstract*

Oder auch Einleitung 🍓 für **VER 2.0.0**

Die  $\text{\LaTeX}$ -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 „2.0.0 - Jake ist auch nur Java“ besitzt den Status Work in Progress!

## Inhaltsverzeichnis

---

<b>1 Einleitung</b>	<b>1</b>
1.1 Installieren von Lilly	1
1.1.1 Linux	
1.1.2 Windows <small>WAR Ausstehend</small>	
1.1.3 MacOS <small>WAR Ausstehend</small>	
1.1.4 Keine Installation	
1.2 Erstellen eines Dokuments mit Lilly <small>VER 2.0.0</small>	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	
<b>2 Mathe</b>	<b>8</b>
2.1 Weitere Befehle	8
2.1.1 Operatoren	
2.1.2 Symbole	
2.1.3 Kompatibilität	
2.1.4 Shortcuts	
2.2 Plots <small>VER 1.0.8</small>	13
2.2.1 graph-Environment	
2.3 3D-Plots	16
<b>3 Grafiken</b>	<b>17</b>
3.1 Grundlegende Symbole	17
3.1.1 Die Ampeln	
3.1.2 Emoticons <small>WAR Ausstehend</small>	
3.1.3 Utility <small>WAR Ausstehend</small>	
3.2 Diagramme & Graphen	18
3.2.1 Graphen	
3.2.2 Rotation	
3.2.3 Automaten <small>WAR Work in Progress</small>	
3.2.4 Schaltkreise <small>WAR Ausstehend</small>	
3.2.5 Neuronen <small>WAR Work in Progress</small>	
3.3 Mitgelieferte Grafiken	22
<b>4 Farben</b>	<b>25</b>
4.1 Die normalen Farbprofile	25
4.1.1 Das Standardfarbprofil	

4.1.2 Das Druckprofil . . . . .	28
4.2 Farberweiterungen . . . . .	28
4.3 Weitere Planungen . . . . .	29
<b>5 Listings</b>	<b>30</b>
5.1 Die grundlegenden Eigenschaften . . . . .	30
5.1.1 Grundlegendes Design	
5.1.2 Das MAIN-Paket	
5.1.3 Das MIPS-Paket	
5.2 Marker und weitere Befehle . . . . .	34
5.2.1 Literates	
5.2.2 Marker	
<b>6 Boxen</b>	<b>36</b>
6.1 Grundlegendes . . . . .	36
6.1.1 Eine kleine Einführung	
6.1.2 Der Box-Controller	
6.2 Die Boxmodi . . . . .	38
6.2.1 Default-Design	
<b>7 Jake</b>	<b>40</b>
7.1 Grundlegendes . . . . .	40
7.1.1 Entwicklung	
7.1.2 Die Installation	
7.1.3 Die Schnittstelle	
<b>8 Aussicht</b>	<b>41</b>
8.1 Bekannte Probleme . . . . .	41
8.1.1 TikZ	
8.2 Todos . . . . .	41
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	
<b>9 Anhang</b>	<b>43</b>
9.1 Version  . . . . .	43
9.1.1 Installation in Linux	
9.1.2 Spezifikation: Plots	
9.2 Version  . . . . .	45
9.2.1 Installation in Linux	
9.2.2 Installation in MacOS	
<b>10 *</b>	<b>47</b>



# 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 gearbeitet.

VER 2.0.0

#### 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](mailto: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 < 2.0.0 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. Einmal installiert lässt sich *Jake* einfach durch `jake` verwenden.

### 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 WAR Ausstehend

## 1.1.3 MacOS WAR Ausstehend

## 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 2.0.0

### 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 unter anderem 4 Optionen:

VER 1.0.7

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

Mit VER 2.0.0 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:

```
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 (*Die Bezeichnung der Listen sind bisher noch inkonsistent :/*). Beispielhaft:

```
1 \listofDEFINITIONS
```

erzeugt hierbei (*Natürlich sind die Linien nur zur Trennung eingefügt.*):

## Alle Definitionen

1	📌 Titel	4
2		4
3	Titel	37
4	📌 Titel	37

### 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 → <sup>5</sup>)

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**.

## 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).

VER 1.0.4

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 1.0.7



### 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
```

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          = @[SELTEFX]
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 T<sub>E</sub>X-Datei:

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

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

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

### Aufgabe 3 – Tolle Aufgabe

Die Aufgabenbeschreibung, blah, blah, blah, ...

- a) Teilaufgabe a)
- b) Teilaufgabe b)
- c) ...

- a) Antwort zu Teilaufgabe a)
- b) Antwort zu Teilaufgabe b)
- c) ...

# 2

## MATHE

EINZELNE VARIATIONEN UND EINE MENGE ABKÜRZUNGEN

VER 2.0.0

An sich ändert LILLY nicht viel an der normalen Implementation der Mathewelt. Dieses Paket liegt hier:

`\LILLYxPATHxMATHS = source/Maths`

### ◇ `\LILLYxMathxMode`

v1.0.3

Der verwendete Mathemodus lässt sich mithilfe des Befehls `\LILLYxMathxMode` frei einstellen. Standardmäßig wird dieser Wert auf *normal* gesetzt.

#### Bemerkung 2.1 – Standalone-Math

Mit `VER 2.0.0` wurde die Mathe-Integration als eigenes Paket (`LIB LILLYxMATH`) etabliert, welches sich auch eigenständig über

```
1 \usepackage{LILLYxMATH}
```

auch ohne das Verwenden der restlichen LILLY-Welt benutzen.

## 2.1 Weitere Befehle

### 2.1.1 Operatoren

Diese Definitionen befinden sich in der Datei: Maths/\_LILLY\_MATHS\_OPERATORS. Sie werden mit `VER 2.0.0` automatisch mit dem Einbinden von `LIB LILLYxMATH` geladen.

#### ◇ `\overbar{text}`

v1.0.3

Lilly liefert den Befehl auf Basis von `mkern` so, dass er direkt Abstände zwischen den Overlines definiert, sodass kein manueller Abstand eingefügt werden muss. So ergibt sich:

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

#### ◇ `\das`, `\sad`, `\daseq`, `\qesad`, `\shouldeq`

v1.0.3

Für Definitionen gibt es die Befehle `\das` ( $:=$ ), `\sad` ( $\equiv$ ), `\daseq` ( $:\Leftrightarrow$ ), `\qesad` ( $\Leftrightarrow$ ) sowie `\shouldeq` ( $\stackrel{!}{=}$ ). All diese Befehle funktionieren sowohl in einer Matheumgebung, das auch im normalen text, sie werden mit `\ensuremath` abgesichert!

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

#### ◇ `\sqrt[n]{math-Ausdruck}`

v1.0.3

Weiter wurde das Aussehen der Wurzel verändert und die Möglichkeit hinzugefügt, über das optionale Argument „n“ höhere Wurzeln zu Formulieren, wir erhalten folgendes:

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

#### ◇ `\det`, `\adj`, `\LH`, `\eig`, `\Dim`, `\sel`, `\sign`, `\diag`, `\LK`, `\rg`, `\KER`, `\Eig`

v1.0.3

Diese vereinfachenden Operatoren solles es ermöglichen Schneller verschiedene mathematische Operatoren zu setzen

◇ <code>\det</code> (det)	◇ <code>\Dim</code> (dim)	◇ <code>\LK</code> (LK)
◇ <code>\adj</code> (adj)	◇ <code>\sel</code> (SEL)	◇ <code>\rg</code> (rg)
◇ <code>\LH</code> ( $\mathcal{LH}$ )	◇ <code>\sign</code> (sign)	◇ <code>\KER</code> (ker)
◇ <code>\eig</code> (Eig)	◇ <code>\diag</code> (diag)	◇ <code>\Eig</code> (Eig)

#### ◇ `\Im`, `\mod`, `\Re`, `\emptyset`

v1.0.2

Auch wurde das Aussehen von `\mod`, `\Im`, `\Re` und `\emptyset` modifiziert:

◇ <code>\mod</code> (MOD)	◇ <code>\Re</code> ( $\Re$ )
◇ <code>\Im</code> ( $\Im$ )	◇ <code>\emptyset</code> ( $\emptyset$ )

#### ◇ `\inf`, `\sup`, `\min`, `\max`

v1.0.6

Auch hierbei handelt es sich wieder um stupide Abbildungen im Operator-Style:

◇ <code>\inf</code> (inf)	◇ <code>\min</code> (min)
◇ <code>\sup</code> (sup)	◇ <code>\max</code> (max)

#### ◇ `\abs{math-Ausdruck}`

v1.0.9

Dieser Befehl vereinfacht das Schreiben von Betragsstrichen. Diese passen sich zudem automatisch an die vertikalen Dimensionen des Ausdrucks an:

<code>\abs{\frac{\pi - x^2}{\log 3x}}</code>	$\frac{\pi - x^2}{\log 3x}$
<code> \frac{\pi - x^2}{\log 3x} </code>	$ \frac{\pi - x^2}{\log 3x} $

#### ◇ `env@matrix[Spaltendefinition]`, `env@pmatrix[Spaltendefinition]`

v1.0.2

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







◇ `\obda`, `\Obda`

v1.0.8

Schreibt entsprechend o.B.d.A. (`\obda`) und O.B.d.A. (`\Obda`) und beschleunigt damit das Tippen in Beweisen ☺.

◇ `\gdw`, `\limn`, `\sumn`, `\limk`, `\sumk`

v1.0.7

Setzt verschiedene mathematische Ausdrücke:

$$\begin{array}{lll} \diamond \text{ \code{\gdw} } (\Leftrightarrow) & \diamond \text{ \code{\sumn} } (\sum_{n=0}^{\infty}) & \diamond \text{ \code{\sumk} } (\sum_{k=0}^{\infty}) \\ \diamond \text{ \code{\limn} } (\lim_{n \rightarrow \infty}) & \diamond \text{ \code{\limk} } (\lim_{k \rightarrow \infty}) & \end{array}$$

◇ `\x[spacing=<~>]`, `\y[spacing=<~>]`, `\z[spacing=<~>]` WAR Veraltet

v1.0.2

Setzt entsprechend:  $x$ ,  $y$  und  $z$

◇ `\ceil[math-Ausdruck]`, `\floor[math-Ausdruck]`

v2.0.0

Verkürzt das Schreiben von: `\left\lfloor``<Ausdruck>``\right\rfloor` beziehungsweise `\lceil` & `\rceil` entsprechend:

$$\diamond \text{ \code{\ceil} } \left(\left\lceil \frac{a}{b} \right\rceil\right) \qquad \diamond \text{ \code{\floor} } \left(\left\lfloor \frac{a}{b} \right\rfloor\right)$$

## 2.2 Plots VER 1.0.8

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

Diese Definitionen befinden sich in der Datei: Maths/\_LILLY\_MATHS\_PLOTS. Sie werden mit VER 2.0.0 automatisch mit dem Einbinden von LIB LILLYxMATH geladen.

◇ `\plotline[Farbe=<Ao>][Variable=<x>]{Term}[offset=<0>]`

v1.0.8

Zeichnet in eine **Graph-Umgebung** eine Funktion (siehe Umgebung für Beispiel). Existiert auch außerhalb von `env@graph`, ist aber hier nur eingeschränkt nutzbar. Mit `offset`<sup>v2.0.0</sup> lässt sich die Funktion entsprechend verschieben.

◇ `\plotseq[Farbe=<Ao>][Variable=<x>]{Term}[Obergrenze=<maxX>][Untergrenze=<1>][Dicke=<1pt>]`

v1.0.8

Zeichnet in eine **Graph-Umgebung** eine Folge zwischen *Unter-* und *Obergrenze* mit Punkten der Größe *Dicke* (siehe Umgebung für Beispiel). Existiert auch außerhalb von `env@graph`, ist aber hier nur eingeschränkt nutzbar.

◇ `\xmark[text=<x>]{PosX}[linelength=<0.15>]`

v2.0.0

Setzt einen Marker auf der  $x$ -Achse bei  $PosX$  mit dem text  $text$ . Für ein Beispiel, siehe **Graph-Umgebung**.

◇ `\ymark[text=<xy>]{PosY}[linelength=<0.15>]`

v2.0.0



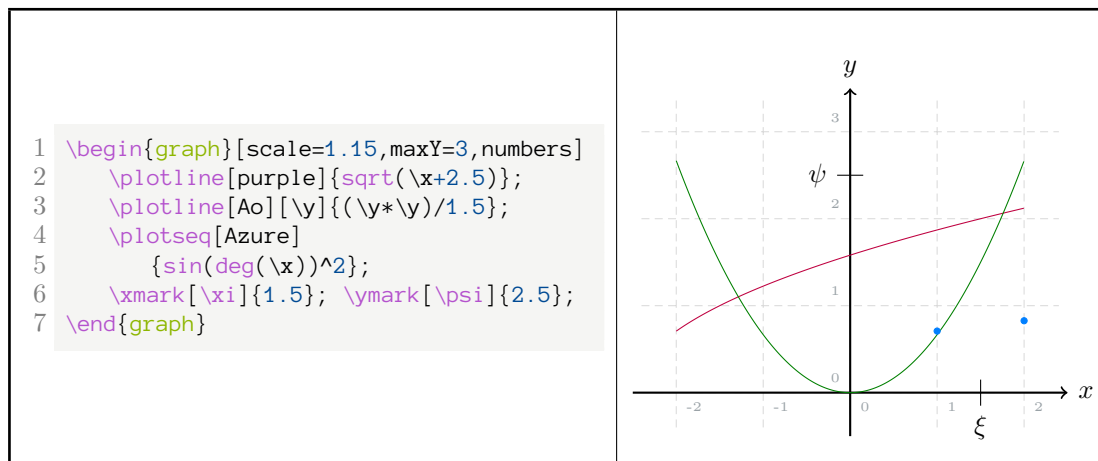
Setzt einen Marker auf der  $y$ -Achse bei  $PosY$  mit dem text  $text$ . Für ein Beispiel, siehe [Graph-Umgebung](#).

## 2.2.1 graph-Environment

◇ `env@graph`[*Konfigurationen*][*Tikz-Argumente*]

v1.0.8

Es existiert die folgende Implementation der Graph-Umgebung:



Für die *Konfigurationen* gibt es folgende Möglichkeiten:

Bezeichner	Typ	Standard	Beschreibung
scale	Zahl	1	Skalierungsfaktor
xscale	Zahl	1	$x$ -Skalierungsfaktor <sup>v2.0.0</sup>
yscale	Zahl	1	$y$ -Skalierungsfaktor <sup>v2.0.0</sup>
minX	Zahl	-2	X-Achse Start
maxX	Zahl	2	X-Achse Ende
minY	Zahl	0	Y-Achse Start
maxY	Zahl	4	Y-Achse Ende
offset	Zahl	0.4	Zusatzlänge Achsen
loffset	Zahl	0.1	Unbeachteter Zusatz Achsen
labelX	String	$\$x\$$	Bezeichner X-Achse
labelY	String	$\$y\$$	Bezeichner Y-Achse
samples	Zahl	250	Anzahl an Kalkulationen
numbers	<>	false	Zeigt Zahlen an
numXMin	Zahl	0	Nummernstart $x$

numYMin	<i>Zahl</i>	0	Nummernstart $y$
numbersize	<i>Zahl</i>	5	Schriftgröße Nummerierung
labelsize	<i>Zahl</i>	10	Schriftgröße Texte

---

◇ `env@egraph`[*Konfigurationen*]

v2.0.0

Funktioniert analog zu `env@egraph`, erlaubt allerdings keine weiteren *Tikz-Argumente*, sondern macht von `env@tikzternal` gebrauch, kann also ausgelagert werden. (TODO: LINK)

◇ `env@wgraph`{*Ausrichtung*}[*Konfigurationen*][*Tikz-Argumente*]  
[*wrapfig-Zusatz*][*width*= $\langle\text{0pt}\rangle$ ]

v1.0.8

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

```

1 \begin{wgraph}{1}[][][\caption{Wichtiger Graph}][400pt]
2 \plotline{\x*\x}
3 \end{wgraph}
```

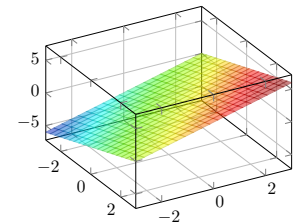
## 2.3 3D-Plots

Bisher sind noch keine Definitionen für 3-Dimensionale Plots integriert. Deswegen hier die exemplarische Definition eines 3D-Plots:

```

1 \begin{tikzternal}[scale=0.5]
2 \begin{axis}[3d box=complete, colormap/bluered,
3             grid=major,view={60}{40},
4             z buffer=sort, data cs=polar]
5     \addplot3[data cs=cart,surf,domain=-3:3,
6               samples=20, opacity=0.5] {x+y};
7 \end{axis}
8 \end{tikzternal}

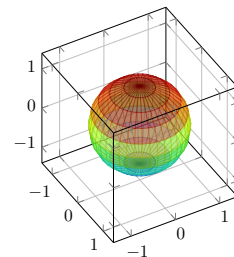
```



```

1 \begin{tikzternal}[scale=0.6]
2 \begin{axis}[3d box=complete, axis equal,
3             image, colormap/bluered,grid=major,view=
4             ={60}{40},z buffer=sort,enlargelimits=0.
5             .2,scale=2.3]
6 \addplot3[%
7     opacity = 0.5, surf,
8     samples = 21, variable = \u,
9     variable y = \v, domain = 0:180,
10    y domain = 0:360,
11    ]
12    ({cos(u)*sin(v)}, {sin(u)*sin(v)},
13     {cos(v)});
14 \end{axis}
15 \end{tikzternal}

```



## 3

## GRAFIKEN

ETLICHE VEREINFACHUNGEN UND ANDERE FREUDEN :D

VER 1.0.2

Dieses Paket liegt hier:

`\LILLYxPATHxGRAPHICS = source/Graphics`**Bemerkung 3.1 – Standalone-Graphics**

Mit `VER 2.0.0` wurde die Grafik-Integration als eigenes Paket (`LIB LILLYxGRAPHICS`) etabliert, welches sich auch eigenständig über

```
1 \usepackage{LILLYxGRAPHICS}
```

auch ohne das Verwenden der restlichen LILLY-Welt benutzen.

## 3.1 Grundlegende Symbole

Diese Definitionen befinden sich in der Datei: `source/Graphics/Tikz-Core/_LILLY-TIKZ_SYMBOLS`. Sie werden mit `VER 2.0.0` automatisch mit dem Einbinden von `LIB LILLYxGRAPHICS` geladen.

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 (<https://www.ctan.org/pkg/pgf>).

### 3.1.1 Die Ampeln

Diese Definitionen befinden sich in der Datei: `../Tikz-Core/_LILLY-TIKZ-AMPELN`. An sich handelt es sich hierbei um ein kleines Shortcut-Sammelsurium für Ampeln:

◇ `\ampelG`, `\ampelY`, `\ampelR`, `\ampelH`

v1.0.2

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

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

### 3.1.2 Emoticons WAR Ausstehend

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

- ◇ `\Ninja` (🥷)
- ◇ `\Smiley` (😊)
- ◇ `\Sadey` (😏)
- ◇ `\Xey` (😏)
- ◇ `\Innocey` (😏)
- ◇ `\Walley` (😏)
- ◇ `\dSadey` (😏)
- ◇ `\Fire` (🔥)
- ◇ `\Autumntree` (🍁)

### 3.1.3 Utility WAR Ausstehend

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

## 3.2 Diagramme & Graphen

### 3.2.1 Graphen

Diese Definitionen befinden sich in der Datei: `source/Graphics/Tikz-Core/_LILLY-TIKZ_GRAPHEN`. Sie werden mit VER 2.0.0 automatisch mit dem Einbinden von LIB LILLYxGRAPHICS geladen.

#### Bemerkung 3.2 – Motivation

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 (<https://www.ctan.org/pkg/pgf>).

- ◇ `\POLYRAD` (length) v1.0.2

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, allen voran die Ur-Funktion:

- ◇ `\graphdot{fill-color}{(PosX,PosY)}{node-name}{border-color}`, v1.0.2  
`\tgraphdot{fill-color}{(PosX,PosY)}{node-name}{border-color}`

Die Befehle unterscheiden sich darin, dass der `\tgraphdot` das Farburgument ignoriert und entsprechend transparent (`fill opacity = 0`) als Füllfarbe verwendet:

<code>\graphdot{DebianRed}{(0,0)}{42}{a}{Azure}</code>	
<code>\tgraphdot{DebianRed}{(0,0)}{42}{a}{Azure}</code>	

- ◇ `\oragraphdot`, `\blugraphdot`, `\gregraphdot`, `\purgraphdot`, `\golgraphdot`, v1.0.2  
`\blagraphdot`, `\nographdot`, `\margraphdot`

Alle weiteren `graphdots` sind nun nichts weiteres als Shortcuts für die eben genannten Befehle und besitzen die Signatur: `\oragraphdot{(PosX,PosY)}{Text}{node-name}`:

- ◇ `\oragraphdot` (🟡)      ◇ `\purgraphdot` (🟣)      ◇ `\nographdot` (⚫)
- ◇ `\blugraphdot` (🟢)      ◇ `\golgraphdot` (🟡)      ◇ `\margraphdot` (🔴)
- ◇ `\gregraphdot` (🟢)      ◇ `\blagraphdot` (⚫)

Zur Information, alle diese Befehle wurden wie folgt präsentiert:

```
1 \tikz{\graphdot}{(0,0)}{42}{a}};
```

wobei `\graphdot` entsprechend ersetzt wurde, weiter wurde für den Textfluss noch die Boxposition angepasst, dies spielt allerdings für den Graphen keine Rolle. Mit VER 2.0.0 wurden die Farben der Dots der neuen Palette entsprechend portiert.

◇ `\graphPOI{(PosX,PosY)}{accent-color}{year}{obj-name}{brief}`  
`{img-path}{img-link}{extra}`

v1.0.4

Präsentiert ein Timeline Point-of-interest, der schnell einen einheitlichen look für Timelines garantiert. Im Folgenden eine repräsentation, die den Wirrwarr an Optionen etwas übersichtlicher macht. Es gilt zu beachten, dass `\extra` hier die Rolle des entsprechenden Landes einnimmt:

<pre>1 \begin{tikzternal}[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{tikzternal}</pre>	<div style="display: flex; justify-content: space-between; align-items: center;"> <div style="text-align: left;"> <p>— 1999 n.Chr.</p> <p><b>Florian Sihler:</b> Florian Sihler ist der Autor dieses Dokuments.</p> </div> <div style="text-align: right;"> <p>Deutschland</p>  </div> </div>
---	--

Hier wurde aus Platzgründen die Größe angepasst. Es gibt auch einen weiteren Befehl der es ermöglicht den `\graphPOI`-Befehl einzuschränken:

◇ `\LILLYxMODExEXTRA`

v1.0.4

Wir dieser Befehl auf `\true` (TRUE) gesetzt, so wird `\graphPOI` so konfiguriert, dass die zugehörige Grafik angezeigt wird. Ist dies nicht der Fall (in anderen Worten: `\LILLYxMODExEXTRA=\false`), so wird kein Bild angezeigt (auch der Link existiert dann nicht). Diese Version wurde erstellt um Urheberrechtsverletzungen zu vermeiden.

◇ `\PgetXY{Point}{out:x-cord}{out:y-cord},`  
`\PgetX{out:x-cord}, \PgetY{out:y-cord}`

v2.0.0

Da es oft notwendig ist die Koordinate eines Punktes weiter zu benutzen und da das Kreuzen von Koordinaten nervig ist, gibt es verschiedene Befehle die es erlauben, die entsprechenden Koordinaten zu speichern, wobei die letzteren beiden nur lesbarere Alternativen für die erste sind, sofern die entsprechend andere Koordinate nicht benötigt wird:

<pre>1 \begin{tikzternal} 2   \node (A) at (1,2) {A}; 3   \PgetXY{(A)}{\myX}{\myY}; 4   % Befehle werden gebunden 5   \node (B) at (\myX,0) {B}; 6   \PgetY{(B)}{\anotherY}; 7   \node (C) at (1.5*\myY,\anotherY) {C}; 8 \end{tikzternal}</pre>	<table border="1" style="width: 100%; height: 100%; border-collapse: collapse; text-align: center;"> <tr> <td style="width: 25%; height: 25px;"></td> <td style="width: 25%; height: 25px;">A</td> <td style="width: 25%; height: 25px;"></td> <td style="width: 25%; height: 25px;"></td> </tr> <tr> <td style="height: 25px;"></td> <td style="height: 25px;"></td> <td style="height: 25px;"></td> <td style="height: 25px;"></td> </tr> <tr> <td style="height: 25px;"></td> <td style="height: 25px;">B</td> <td style="height: 25px;"></td> <td style="height: 25px;">C</td> </tr> <tr> <td style="height: 25px;"></td> <td style="height: 25px;"></td> <td style="height: 25px;"></td> <td style="height: 25px;"></td> </tr> </table>		A								B		C				
	A																
	B		C														

Was hierbei auch interessant ist: die Skalierung von  $X$ - und  $Y$ -Koordinaten wird unabhängig voneinander getroffen, das heißt die  $Y$ -Koordinate eines Punktes als die  $X$ -Koordinate eines anderen zu verwenden funktioniert (meist) nicht ohne mathematische Operationen. Das Gitter wurde natürlich nachträglich hinzugefügt:

```
1 \draw[thin,xshift=0.5cm,yshift=0.5cm] (-1,-2) grid (3,2);
```

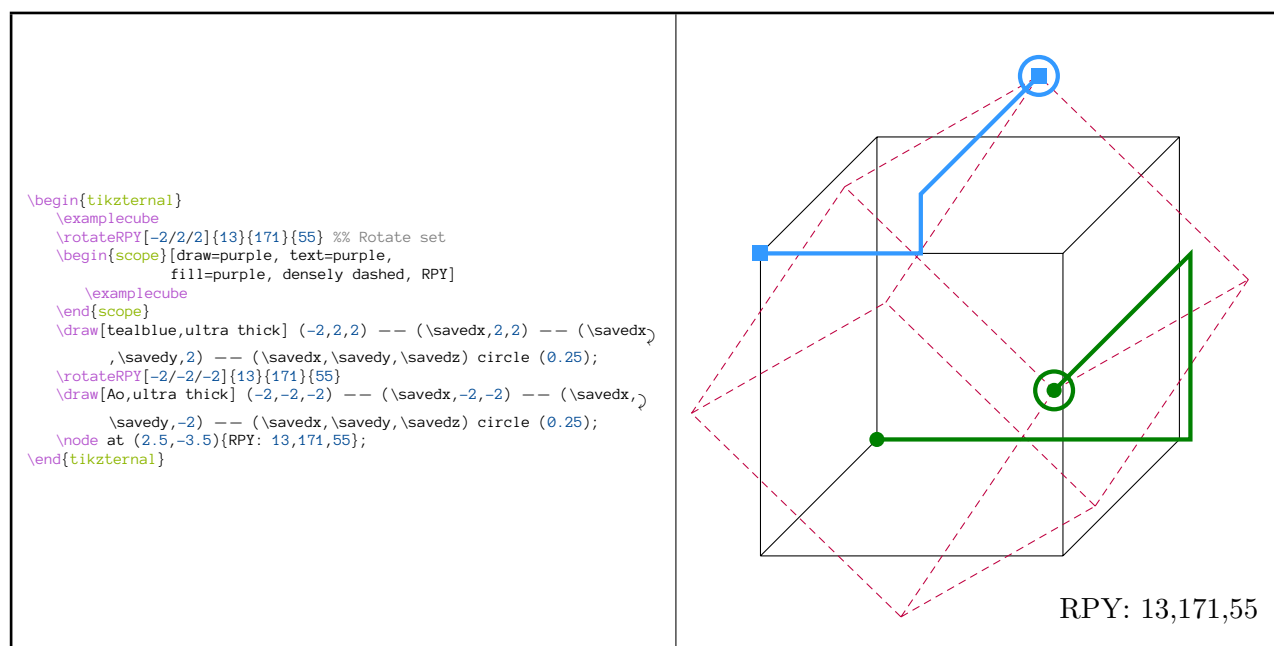
### 3.2.2 Rotation

Diese Definitionen befinden sich in der Datei: `source/Graphics/Tikz-Core/_LILLY-TIKZ_ROTATION`. Sie werden mit `VER 2.0.0` automatisch mit dem Einbinden von `LIB LILLYxGRAPHICS` geladen.

◇ `\rotateRPY[transform-point=<0/0/0>]{roll}{pitch}{yaw}`

v1.0.4

Dieser Befehl wird verwendet um erstellte TikZ Grafiken zu drehen und dementsprechend anzupassen. Dieser Code entstammt der Feder von David Carlisle und Tom Bombadil<sup>(a)</sup> und wird hier beispielhaft illustriert:



### 3.2.3 Automaten WAR Work in Progress

Diese Definitionen befinden sich in der Datei: `source/Graphics/Tikz-Core/_LILLY-TIKZ_AUTOMATEN`. Sie werden mit `VER 2.0.0` automatisch mit dem Einbinden von `LIB LILLYxGRAPHICS` geladen.

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 lautet:

◇ `\loopTo[looseness=<1>]{arc}{node-name}{Text}{Orientierung}`

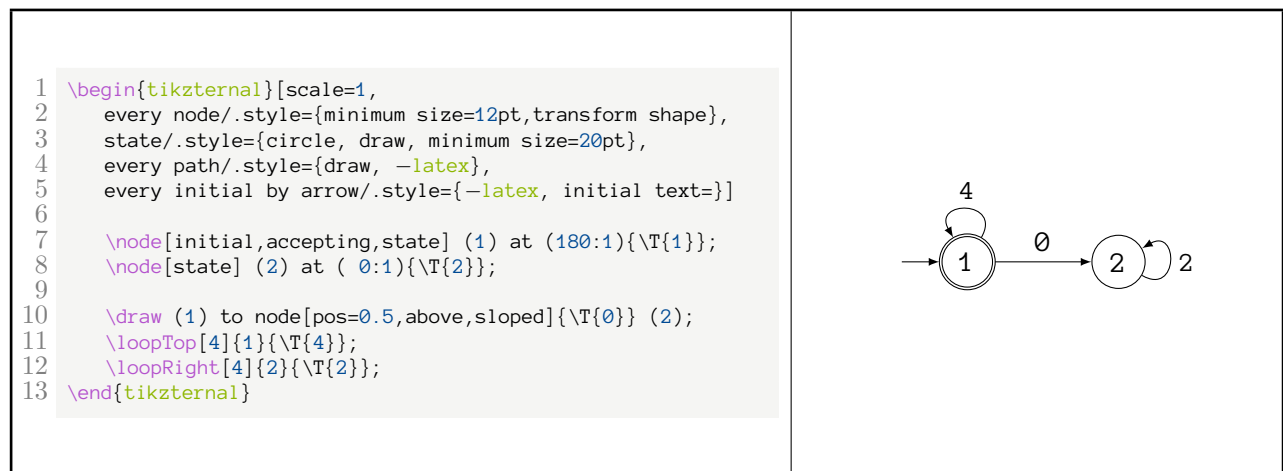
v1.0.3

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

Dieser Befehl setzt grundlegend einen Pfeil, der von einem Knoten aus wieder zu sich selbst führt. Im folgenden sind 4 verschiedene Shortcuts, die für die klassischen Himmelsrichtungen die Pfeile vordefinieren:

◇ `\loopTop[looseness=<1>]{node-name}{Text}, \loopRight[lsns=<1>]{node-name}{Text}, \loopLeft[lsns=<1>]{node-name}{Text}, \loopBot[lsns=<1>]{node-name}{Text}` v1.0.3

Im folgenden sei eine beispielhafte Verwendung gezeigt (der Automat muss keinen Sinn ergeben es soll lediglich die Nutzung verdeutlicht werden):



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 `\state` und `\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, dies wird allerdings erst auf das Bedürfnis hin übernommen.

### 3.2.4 Schaltkreise WAR Ausstehend

### 3.2.5 Neuronen WAR Work in Progress

Diese Definitionen befinden sich in der Datei: `source/Graphics/Tikz-Core/_LILLY-TIKZ_NEURONS`. Sie werden mit VER 2.0.0 automatisch mit dem Einbinden von



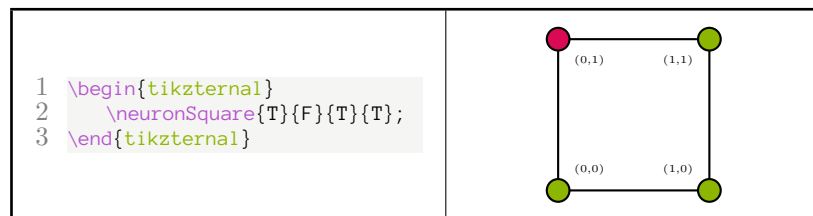
 geladen.

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. *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.*

◇ `\neuronSquare{pos:00}{pos:01}{pos:10}{pos:11}`


v1.0.5

Es wurde bisher auch nur durch das Bereitstellen eines einzelnen Befehls implementiert: `\neuronSquare`. Dieser funktioniert seinerseits lediglich in einer *tikzpicture/tikzternal* 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:




Hierbei steht ein T (true) natürlich für einen akzeptierten, ein F (false) entsprechend für einen nicht akzeptierten Befehl. Aktuell ist geplant, dass der Befehl auch für 1-, 3- und 4-dimensionale Räume eine Option anbietet (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 möglich sein über ein optionales Argument die Position (relativ) zu bestimmen!

## 3.3 Mitgelieferte Grafiken

Diese Definitionen befinden sich in der Datei: `source/Graphics/LILLYxGRAPHICSxPROVIDER.sty`. Sie werden mit  automatisch mit dem Einbinden von

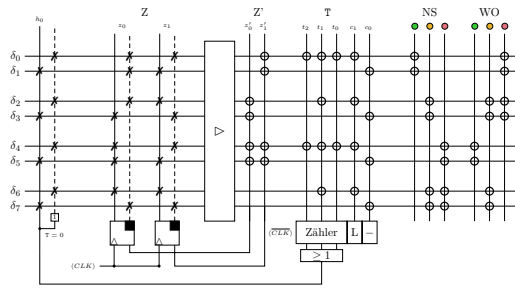
 geladen.

Dieser Teil existiert weiter auch als eigenes Paket mit:  und hängt vom Mutterpaket ab.

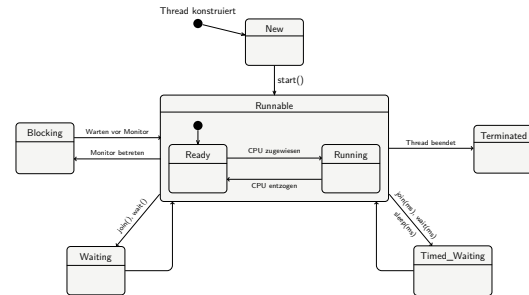
◇ `\getGraphics[width=<\linewidth>]{path}[height]`

v2.0.0

Erlaubt den Zugriff auf zahlreiche Grafiken, die im Rahmen der Arbeit entstanden sind. Bei einer Angabe von Breite und Höhe gewinnt die Breite, da stets nur eine Dimension skaliert wird! Die bisher enthaltenen Grafiken können durch `jake get` abgerufen werden:



\getGraphics{Rechner/PLAAmpel}



\getGraphics{Software/ThreadState}

Die Größe skaliert sich in der Regel automatisch, allerdings existieren auch Grafiken, die automatisch nicht skaliert werden, da sie Code oder andere nicht skalierfähige Elemente enthalten:

Liefert Metadaten zur Datei, *xml* ist das einzige reservierte Keyword

Wurzelement →

Kind des „blaetter“-Elements →

```

<?xml version="1.0" encoding="UTF-8"?>
  <!-- Alle Blaetter -->
  <blatt num="42" abgabe="10.10.2019">
    <aufgabe topic="Java_NIO" points="12">
      <unteraufgabe nr="1.1">
        „punkte“-Tag → <punkte>4</punkte> ← „punkte“-Endtag
        <text>Bitte ... &amp;</text>
        <grafik/> ← Leerer „Grafik“-Tag
      </unteraufgabe>
    </aufgabe>
  </blatt>
</blaetter>
  
```

Verbosere End-Tag-Spaß

\getGraphics{Software/XMLUebersicht}

#### ◇ \getGraphicsPath{path}

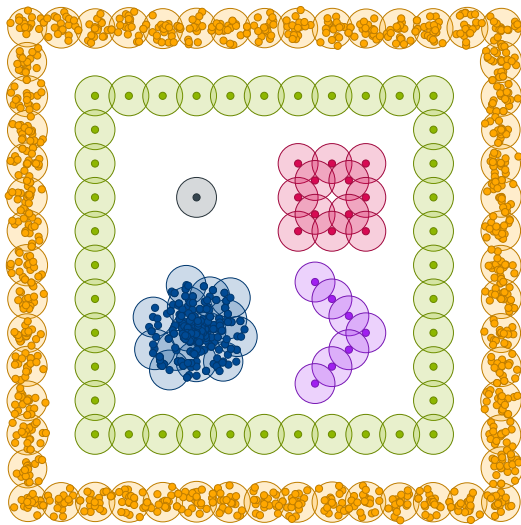
v2.0.0

Liefert den absoluten Pfad zu einer Grafik. Beispiel:  
 \getGraphicsPath{Software/XMLUebersicht} liefert:  
 source/Data/Graphics/Software/XMLUebersicht.

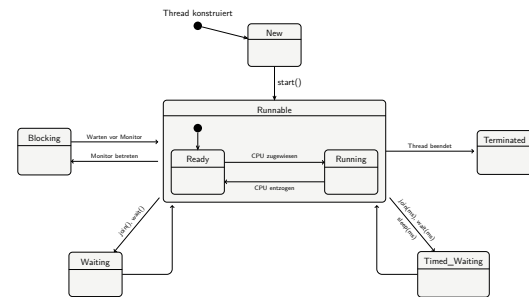
#### ◇ \getPrerendered[width=(\linewidth)][path][height]

v2.0.0

Erlaubt eine automatisch an die Seitenbreite skalierte Implementation von bereits vorberechneten Grafiken. Bei einer Angabe von Breite und Höhe gewinnt die Breite, da stets nur eine Dimension skaliert wird! Sie werden in der Grafiksammlung durch den Tag pdf gekennzeichnet (die Breite wurde im Beispiel angepasst, oder lassen sich durch das Anfügen eines „-pdf“-Suffix:



\getPrerendered{Eigene/Proseminar/Cluster/rolf-special}



\getPrerendered{Software/ThreadState-pdf}

*Es gilt zu beachten, dass die bereits vorgenerierten Grafiken von den manuell generierten abweichen können!*

## 4

## FARBEN

VIELE VIELE BUNTE FARBEN

VER 1.0.4

Damit die verwendeten Farben, je nach Profil und Wunsch in Paletten gruppiert gesetzt werden können, wurde dieses Paket ins Leben gerufen. Es befindet sich hier:

`\LILLYxPATHxDATA/Colors = source/Data/Colors`

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 nicht mehr die Design-Unabhängigkeit von LILLY garantiert ist!**

### Bemerkung 4.1 – Standalone Color

Mit VER 2.0.0 wurde die Farben-Integration als eigenes Paket (LIB LILLYxCOLOR) etabliert, welches sich auch eigenständig über

```
1 \usepackage{LILLYxCOLOR}
```

auch ohne das Verwenden der restlichen LILLY-Welt benutzen.

## 4.1 Die normalen Farbprofile

Mit VER 2.0.0 werden die Hauptfarben generell mit diesem Paket zur Verfügung gestellt, während die Profile und Erweiterungen sich mit den Mappings befassen, dieser Prozess ist noch im Gange und natürlich wäre es Wünschenswert, wenn alle Farben über ein entsprechendes Mapping gesetzt werden.

Mit dem Paket LIB LILLYxLIST in VER 2.0.0, wurden die zur Verfügung stehenden Farben in Listen Organisiert:

- ◇ `\LISTxColors` (Quelle: LillyColorList)
- ◇ `\LISTxCompatColors` (Quelle: LillyCompatColorList)

Sie halten die jeweiligen Farben nach dem Schema: Name/R/G/B und können so entsprechend auch manipuliert werden. Die Farben können jeweils über folgenden Befehl Lilly gegenüber Registriert werden:

◇ `\registerColors{Liste:n/r/g/b}{Name}, \updateColors{Liste:n/r/g/b}{Name}` v2.0.0

Dieser Befehl definiert die neuen Farben einmal mittels `\providecolor` (register) und mit `\definecolor` (update). Die Listen-Signatur entspricht: Name der Farbe/R-Wert, /G-Wert/B-Wert. Da die Farben „nur“ registriert werden, kann man sie von außerhalb

überschreiben, was allerdings zunichte gemacht wird, sofern man sie mittels `\updateColors` innerhalb des Dokuments überschreibt. Bisher sieht Lilly eine derartige Verwendung des Befehls nicht vor, er wird also intern nirgendwo verwendet.

In Lilly findet das registrieren der Farben wie folgt statt:










```
1 \storeLillyColorList{LISTxColors}
2 \registerColors{\LISTxColors}{}
3 \storeLillyCompatColorList{LISTxCompatColors}
4 \registerColors{\LISTxCompatColors}{Compat-}
```

Hier eine Auflistung der Standartfarben in `\LISTxColors`:

 Butter (r: 255, g: 247, b: 155)	 bondiBlue (r: 0, g: 149, b: 182)
 Aureolin (r: 253, g: 238, b: 0)	 antiVeg (r: 190, g: 238, b: 239)
 Amber (r: 255, g: 191, b: 0)	 DarkOrchid (r: 104, g: 34, b: 139)
 ChromeYellow (r: 255, g: 167, b: 0)	 Veronica (r: 160, g: 32, b: 240)
 DarkChromeYellow (r: 255, g: 140, b: 0)	 Orchid (r: 180, g: 82, b: 205)
 Coquelicot (r: 255, g: 56, b: 0)	 Amethyst (r: 153, g: 102, b: 204)
 Cinnabar (r: 227, g: 66, b: 52)	 AntiqueFuchsia (r: 145, g: 92, b: 131)
 BrightMaroon (r: 195, g: 33, b: 72)	 BritishRacingGreen (r: 0, g: 66, b: 37)
 Cherry (r: 222, g: 49, b: 99)	 DatmouthGreen (r: 0, g: 105, b: 62)
 AlizarinCrimson (r: 227, g: 28, b: 54)	 Ao (r: 0, g: 128, b: 0)
 Amaranth (r: 229, g: 43, b: 80)	 Leaf (r: 44, g: 171, b: 63)
 AmericanRose (r: 255, g: 3, b: 62)	 AppleGreen (r: 141, g: 182, b: 0)
 Awesome (r: 255, g: 32, b: 82)	 BrightGreen (r: 102, g: 255, b: 0)
 BrightPink (r: 255, g: 0, b: 127)	 MudWhite (r: 245, g: 245, b: 243)
 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: 74, b: 148)	 Charcoal (r: 54, g: 69, b: 79)
 Azure (r: 0, g: 127, b: 255)	

## Bemerkung 4.2 – Kompatibilität

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

 gold (r: 255, g: 215, b: 50)	 beauty (r: 104, g: 55, b: 107)
 dgold (r: 232, g: 177, b: 38)	 dpurple (r: 86, g: 60, b: 92)
 mint (r: 255, g: 128, b: 0)	 limegreen (r: 51, g: 204, b: 51)
 dorange (r: 255, g: 102, b: 0)	 skyblue (r: 60, g: 179, b: 113)
 thered (r: 255, g: 47, b: 47)	 tealblue (r: 51, g: 153, b: 255)
 candypink (r: 227, g: 112, b: 122)	 superlightgray (r: 240, g: 240, b: 240)
 ddpurple (r: 128, g: 0, b: 128)	

Sie sollten nicht mehr verwendet werden!

#### ◇ `\Hcolor`, `\HBColor`

v1.0.9

Diese Farben können mithilfe von *Jake* auch durch den Parameter `lilly-signatur-farbe` gesetzt werden, wobei `\HBColor` immer eine etwas dunklere Variante der Farbe darstellt. Standardmäßig ist diese Farbe Leaf (●).

#### ◇ `\LillyxStorexCurrentColorProfile`, `\LillyxRestorexCurrentColorProfile`

v2.0.0

Diese Befehle speichern das aktuelle Farbprofil und Laden es entsprechend wieder. Diese Mechanik wurde zum Beispiel hier verwendet um dynamisch die entsprechenden Farbprofile (wie das *Druckprofil*) anzuzeigen.

### 4.1.1 Das Standardfarbprofil

Diese Definitionen befinden sich in der Datei: `source/Data/Colors/_LILLY_DEFAULT_COLORPROFILE`. Sie werden mit `VER 2.0.0` automatisch mit dem Einbinden von `LIB LILLYxGRAPHICS` geladen.

#### ◇ `\LILLYxColorxInject`

v1.0.1

Dieses Farbprofil wird nur geladen, wenn die Variable `\LILLYxColorxInject` **nicht** definiert ist.

Dieses Farbprofil definiert 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. Wichtig ist, dass mit `VER 2.0.0` auch hier alle Farben jeweils in eine Liste geladen werden. Diese trägt den Namen `LillyProfileColors` (der Zugriff erfolgt wieder über: `\LISTxProfileColors`) und trägt die Verantwortung für die Konstruierten Farben. Lilly kümmert sich bisher noch nicht darum, dass nur gültige Farben in diese Liste gelangen, dies sollte allerdings nur eine untergeordnete Rolle spielen, da andere Farben schlicht ignoriert werden. Alle folgenden Farben werden durch das Präfix `LILLYxColorx` angeführt.

● Definition ( <i>DebianRed</i> )	● Übungsaufgabe ( <i>Veronica</i> )
● Satz ( <i>Ao</i> )	● Zusatzübung ( <i>Veronica</i> )
● Beweis ( <i>DarkMidnightBlue</i> )	● <code>LINKSxMainColor</code> ( <i>DebianRed!85!black</i> )
● Lemma ( <i>DarkMidnightBlue</i> )	● <code>LINKSxMainColorDarker</code> ( <i>DebianRed!75!black</i> )
● Bemerkung ( <i>Charcoal</i> )	● <code>LINKSxCiteColor</code> ( <i>DarkMidnightBlue</i> )
● Zusammenfassung ( <i>ChromeYellow</i> )	● <code>LINKSxUrlColor</code> ( <i>DarkMidnightBlue</i> )
● Beispiel ( <i>Aureolin</i> )	● <code>TITLExCOLOR</code> ( <i>DebianRed!85!black</i> )

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

Beispielhaft lässt sich die Definitionsfarbe mit: `\LILLYxColorxDefinition` abfragen (●). Aus Flexibilitätsgründen wurden alle diese Farben als Befehle implementiert, um sie von den statischen Farben zu unterscheiden.

## 4.1.2 Das Druckprofil

Diese Definitionen befinden sich in der Datei: `source/Data/Colors/_LILLY_PRINT_COLORPROFILE`. Sie werden mit `VER 2.0.0` automatisch mit dem Einbinden von `LIB LILLYxGRAPHICS` bereitgestellt und durch das setzen des Druckmodus geladen.

Auch dieses Profil definiert seine Farben nur, wenn `\LILLYxColorxInject` nicht definiert ist! Die Präsentation der Farben erfolgt wieder mithilfe von: `\LISTxProfileColors`:

● Definition ( <i>DebianRed</i> )	● Definition ( <i>DebianRed</i> )
● Satz ( <i>Ao</i> )	● Satz ( <i>Charcoal</i> )
● Beweis ( <i>DarkMidnightBlue</i> )	● Beweis ( <i>Charcoal</i> )
● Lemma ( <i>DarkMidnightBlue</i> )	● Lemma ( <i>Charcoal</i> )
● Bemerkung ( <i>Charcoal</i> )	● Bemerkung ( <i>Charcoal</i> )
● Zusammenfassung ( <i>ChromeYellow</i> )	● Zusammenfassung ( <i>ChromeYellow</i> )
● Beispiel ( <i>Aureolin</i> )	● Beispiel ( <i>Charcoal</i> )
● Uebungsaufgabe ( <i>Veronica</i> )	● Uebungsaufgabe ( <i>Charcoal</i> )
● Zusatzuebung ( <i>Veronica</i> )	● Zusatzuebung ( <i>Charcoal</i> )
● LINKSxMainColor ( <i>DebianRed!85!black</i> )	● LINKSxMainColor ( <i>Charcoal</i> )
● LINKSxMainColorDarker ( <i>DebianRed!75!black</i> )	● LINKSxMainColorDarker ( <i>Charcoal!90!black</i> )
● LINKSxCiteColor ( <i>DarkMidnightBlue</i> )	● LINKSxCiteColor ( <i>Charcoal</i> )
● LINKSxUrlColor ( <i>DarkMidnightBlue</i> )	● LINKSxUrlColor ( <i>Charcoal</i> )
● TITLExCOLOR ( <i>DebianRed!85!black</i> )	● TITLExCOLOR ( <i>DebianRed</i> )

Die Farbe `\LILLYxColorxLINKSxMainColorDarker` (●) wird hier mithilfe von: `\LILLYxColorxLINKSxMainColor!95!black` generiert.

Eine weitere Repräsentation der Farben ergibt sich durch `\LILLYxCOLORxRainbow` (entstammt den Shortcuts [LINK]):



## 4.2 Farberweiterungen

Es gibt eine Reihe an Farberweiterungen, die die oben definierten Druckprofile hinsichtlich einer gewissen Farbprägung abändern. Die von Lilly standartmäßig inkludierten Profile finden sich hier: `\LILLYxPATHxDATA/Colors/Extensions`:

Bezeichner	Farben
GREEN	
PURPLE	
CHESS	
VOID	

Die Farbprofile können durch das Setzen von `\LILLYxCOLORxEXTENSION` auf den jeweiligen Bezeichner geladen werden.

## 4.3 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.



## 5

## LISTINGS

IST THIS... THE MATRIX?

VER 1.0.0

Zum Setzen von Programmtexten innerhalb von Latexdokumenten stellt dieses Paket eine große Ansammlung verschiedener Sprachen und Dialekten zur Verfügung. Es befindet sich hier:

$$\backslash\text{LILLYxPATHxLISTINGS} = \text{source/Data}$$

### Bemerkung 5.1 – Standalone Listings

Mit VER 2.0.0 wurde die Listings-Integration als eigenes Paket (LIB LILLYxLISTINGS) etabliert, welches sich auch eigenständig über

```
1 \usepackage{LILLYxLISTINGS}
```

auch ohne das Verwenden der restlichen LILLY-Welt benutzen.

*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  $\backslash\text{mipsADD}$  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:  $\backslash\text{LILLYxPATHxLISTINGS}/\text{LILLYxLISTINGS}$ . Sie werden mit VER 2.0.0 automatisch mit dem Einbinden von LIB LILLYxLISTINGS geladen.

### Bemerkung 5.2 – Verwendetes Paket

LILLY verwendet nicht das normale listings-Paket, sondern greift auf das erweiterte Paket listingsutf8 zu, sofern dieses Vorhanden ist. Es werden weiter Definitionen für alle Umlaute gesetzt, sowie eine Reihe an weiteren Ersetzungsregeln. Darunter fällt übrigens auch das Markieren von Zahlen.

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

```
1 \input{\LILLYxPATHxLISTINGS/Packages/_LILLY_PACK_\LILLYxListingsxPACK}
```

Standardmäßig wird so das **MAIN**-Paket geladen, welches alle hier definierten Sprachen mitliefert. Damit die zur Verfügung stehenden Sprachen auch verwaltet werden können, läuft die Verwaltung der Sprachen wieder über eine Liste. Die Liste *RegisteredLanguages* verwaltet hierbei die Registrierten Sprachen (in der Signatur *Sprache/Sprachbezeichner*) und stellt für jede Sprache einen Shortcut zur Verfügung:

- ◇ `\c<Sprache>\{<Code>\}` Setzt den Code mit grauem Hintegrund. Zeilenumbrüche werden hier zwar durchgeführt, allerdings in der Regel nicht optimal gesetzt. Beispiel:

```
1  \c.java{public static void main(String[] args)}
```

Liefert: `public static void main(String[] args)`

- ◇ `\b<Sprache>\{<Code>\}` Setzt den Code farbig auf dem vorhandenen Hintergrund. Beispiel:

```
1  \bjava{public static void main(String[] args)}
```

Liefert: `public static void main(String[] args)`

- ◇ `\i<Sprache>\{<Datei>\}` Lädt und setzt den Programmcode aus der entsprechenden Datei.

Ferner *kann* eine Sprache eine gleichnamige Umgebung mitliefern, die das direkte Setzen des Codes innerhalb eines Textlaufes ermöglicht. Beispiel:

```
1 \begin{java}
2 public class SuperKlass {
3     public static void main(String[] args) {
4         System.out.println("Hallo Welt");
5     }
6 }
7 \end{java}
```

Ergibt:

```
1 public class SuperKlass {
2     public static void main(String[] args) {
3         System.out.println("Hallo_Welt");
4     }
5 }
```

- ◇ `\isLanguageLoaded{LanguageSignature}`

v2.0.0

Prüft ob eine Sprache geladen ist. Als Argument wird hierbei die volle Sprachsignatur erwartet (Sprache/Sprachbezeichner) um auch doppelten Bezeichnern vorzubeugen.

- ◇ `\isLanguageNameLoaded{LanguageName}`

v2.0.0

Prüft ob eine Sprache geladen ist. Als Argument wird hierbei die volle Sprache erwartet, was doppelte Bezeichner natürlich ausschließt, allerdings in den meisten Fällen auch einfacher ist:

```

1 \isLanguageNameLoaded{java} % → TRUE
2 \isLanguageNameLoaded{waffel} % → FALSE

```

#### ◇ `\cmdshowcase{command}`

v2.0.0

Kleiner Shortcut um auch den Inhalt eines Befehls als Listing zu setzen. Betrachte folgendes Beispiel:

```

1 \begin{multicols}{3}
2   \begin{ditemize}
3     \foreach \x in {public,static,void} {
4       \item \cjava{\x} vs. \cmdshowcase[language=lJava]{\x}
5       \newline
6     }
7   \end{ditemize}
8 \end{multicols}

```

Ergibt:

◇ **x** vs. **public**

◇ **x** vs. **static**

◇ **x** vs. **void**

#### ◇ `\LILLYxwriteLst[lstArgs]{Code}` WAR Veraltet

v1.0.8

Setzt Programmcode entsprechend veralteter Definitionen.

### Bemerkung 5.3 – Zugriff auf die eigentliche Sprachdefinition

Um keine Doppeldeutigkeit bezüglich der Sprachen zu erhalten werden alle LILLY-Sprachen durch das „l“-Prefix angeführt. So heißt es nicht „java“ sondern „lJava“, sofern die Sprache manuell geladen werden soll.

## 5.1.2 Das MAIN-Paket

Diese Definitionen befinden sich in der Datei: `\LILLYxPATHxLISTINGS/Packages/_-LILLY_PACK_MAIN`. Sie werden mit VER 2.0.0 automatisch mit dem Einbinden von LIB `LILLYxLISTINGS` geladen.

Nebst der Sprach-Umgebungen werden hier zwei Umgebungen definiert, die dem Stil des Main-Pakets entgegenarbeiten:

#### ◇ `env@lstplain[lstArgs]`, `env@lstnonum[lstArgs]`

v1.0.9

Während erstere einfach nur Code ohne anderweitige Formatierungen setzt, entfernt letztere nur die Aufzählung entsprechender Zahlen:

```

1 \begin{lstplain}[language=lJava]
2 public static void main(String[] args) {
3   System.out.println("Hallo Welt");
4 }
5 \end{lstplain}
6 % Sowie:
7 \begin{lstnonum}[language=lJava]

```

```
8 public static void main(String[] args) {
9     System.out.println("Hallo Welt");
10 }
11 \end{lstnonum}
```

Ergibt:

```
public static void main(String[] args) {
    System.out.println("Hallo_Welt");
}
```

Sowie:

```
public static void main(String[] args) {
    System.out.println("Hallo_Welt");
}
```

### Bemerkung 5.4 – Geladene Sprachen

Hier eine Auflistung aller Sprachen, die über das Main-Paket geladen werden:

◇ assembler	◇ latex	◇ sql	◇ haskell
◇ pseudo	◇ gepard	◇ xsl	◇ cpp
◇ mips	◇ java	◇ chr	◇ python
◇ bash	◇ xml	◇ prolog	◇ json

Widmen wir uns nun allerdings einmal den weiteren Definitionen des MAIN-Pakets: Die allgemeine TypeWriter-Schriftart wird mithilfe von `\LILLYxlstTypeWriter` auf AnonymousPro gesetzt (*Sie wird auch hier für die Dokumentation verwendet*). Zudem lädt MAIN noch das Paket MIPS, auf welches weiter unten noch weiter eingegangen wird. Weiter wird `\lst@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: `\LILLYxPATHxLISTINGS/Languages/_LILLY_LANG_MIPS`. Sie werden mit `VER 2.0.0` automatisch mit dem Einbinden von `LIB LILLYxLISTINGS` geladen.

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.

- ◇ \gitRAW, \git WAR Veraltet

v1.0.0

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/](https://www.github.com/EagleoutIce/MIPS_UniUlm_Examples/): ).

Ursprünglich waren diese Definitionen nur für *Grundlagen der Rechnerarchitektur* gedacht und sollten auch schleunigst wieder dorthin verschwinden (TODO!!)

Es werden einige weitere Stile definiert:

## MIPS

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

● Zeichenketten ( <i>candypink</i> )	● Spezielle Befehle ( <i>limegreen</i> )
● Befehle ( <i>purple</i> )	● Buzzwords ( <i>thered</i> )
● Register ( <i>tealblue</i> )	● Daten-Direktiven ( <i>tealblue!60!black</i> )
● Direktiven ( <i>dgold</i> )	

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

## MIPSSNIP

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

### Bemerkung 5.5 – MIPS

Das gesamte Mipspaket ist seit VER 1.8.0 überholt und bedarf einiger Aufarbeitung, dennoch tut es seinen Dienst für die bisher existenten MIPS-Codes allerdings stehen weitere Besonderheiten wie zum Beispiel Literates nebst der anfänglich implementierten nicht zur Verfügung...

## 5.2 Marker und weitere Befehle

### 5.2.1 Literates

Im Kontext verschiedener Programmiersprachen kam bald der Wunsch auf verschiedene Symbole entsprechend einfach setzen zu können. Bisher werden alle diese Ersetzungsregeln über das Einbinden von LIB LILLYxLISTINGS geladen und ermöglichen es, neben Umlauten auch Symbole einzubinden. Die Ersetzungsregeln werden nicht über eine Liste gehandhabt und sind ebenso vielfältig wie es die Bedürfnisse erfordern. Im Folgenden eine Auflistung aller in VER 2.0.0 enthaltener Ersetzungsregeln:

:bs: „\“	:space: „ “	:ldots: „...“	:yields: „→“
:bmath: „\$“	:ws: „ “	:c: „“	:lan: „⟨“
:emath: „\$“	:cdots: „...“	:float: „f“	:ran: „⟩“
:dollar: „\$“	:cdot: „·“	:exp: „e“	:bcmd: „\“

### 5.2.2 Marker

Mit VER 2.0.0 im Anfangsstadium befinden sich die jeweiligen Marker die es erlauben Fehler oder ganz Allgemein Code-stellen zu markieren, oder von Highlighting zu befreien:

```
1 import java.util.ArrayList;
2
3 public class Example {
4     public static void main(String[] args) {
5         System.out.println("Hallo_Welt");
6     }
7 }
```

```
6         if(args==null)
7             System.out.println("wau");
8     }
9 }
```

```
1 \begin{java}
2 |info|import java.util.ArrayList;|info|
3
4 |plain|public class Example {|plain|
5     public static void main(String[] args) {
6         System.out.|err|PrintLn|err|("Hallo Welt");
7         if(|warn|args==null|warn|)
8             System.out.println("wau");
9     }
10 }
11 \end{java}
```

# 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 bis VER 1.7.0 hierfür lautet:

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

Mit VER 2.0.0 regelt *Jake* die jeweilige Variante und erlaubt es sogar, mehrere Boxmodi gleichzeitig generieren zu lassen:

```
1 jake <Datei> -lilly-boxes: "<Namen:ran>"
```

Um eine Fassng für jede Box zu generieren entspräche das:

```
1 jake <Datei> -lilly-boxes: "DEFAULT.ALTERNATE.LIMERENCE"
```

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.

#### 6.1.2 Der Box-Controller

Diese Definitionen befinden sich in der Datei: Controllers/LILLYxCONTROLLERxBOX. Sie werden mit VER 2.0.0 automatisch mit dem Einbinden von LIB LILLYxCONTROLLERxBOX geladen.

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

\LILLYxBOXxMODE

\LILLYxBOXx\*xLock

`\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` auf `section` das Nummerieren der Box auf die Sektionen festlegen (`TRUE` für ungebunden). Weiter definiert es die folgenden Box-Environments:

`\LILLYxBOXx*xEnable`**Definition 3 – Titel**

moin

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

**Beweis 6.1 – Titel**

moin

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

**Definition 4 – Titel**

moin

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

**Lemma 6.1 – Titel**

moin

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

**Bemerkung 6.1 – Titel**

moin

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

**Zusammenfassung 6.1 – Titel**

moin

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

**Beispiel 6.1 – Titel**

moin

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

**Aufgabe 4 – Titel**

moin

```
1 \begin{aufgabe}{Titel}{3}
2     moin
3 \end{aufgabe}
```

**Satz 6.4 – Titel**

moin

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

Nicht richtig darstellbar aber weiter existiert:

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

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

`\LILLYxBOXx*xBox`



Bisher unterstützt werden:

- |             |           |               |
|-------------|-----------|---------------|
| ◇ Bemerkung | ◇ Beweis  | ◇ Übungsblatt |
| ◇ Beispiel  | ◇ Aufgabe |               |

Setzt man den Wert auf FALSE so wird das sogenannte plain-Design angewendet, welches jedes Design wieder selbst definieren kann!.

Zudem existieren aus Kompatibilitätsgründen auch noch die alten Befehle aus dem `eagleStudiPackage`: `\DEF(2)`, `\BEM(2)`, `task,...`

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*`)

```

\DEF
\BEM
env: task
\inputUB
\inputUBS

```

### Bemerkung 6.2 – zugriff auf die Boxzähler

## 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 \tcbsset{LillyxB0XxDesignxDefault/.style={enhanced jigsaw, pad before 0pt,
2   break*=2mm %
3   pad after break=2mm, lines before break=4, before skip=0pt,
4   boxrule = 0mm, toprule=0.5mm,%
5   bottomtitle=0.5mm,bottomrule=1.2mm, after skip=0pt, enlarge top 0pt,
6   by=\baselineskip,%
7   enlarge bottom by=\baselineskip, sharp corners=south, enforce 0pt,
8   breakable}%
9 }

```

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!
5     white,%
6     colframe=\LILLYxColorxDefinition, #3,%
7     title={%
8       \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*=1mm-\
4      \tcboxedtitleheight},%
5    varwidth boxed title*=-3cm,%
6    boxed title style={
7      frame code={
8        \path[fill=white!30!black]%
9          ([yshift=-1mm,xshift=-1mm]frame.north west)%
10         arc[start angle=0,end angle=180,radius=1mm]%
11         ([yshift=-1mm,xshift=1mm]frame.north east)%
12         arc[start angle=180,end angle=0,radius=1mm];
13        \path[left color=white!40!black,right color=white!40!black,
14              middle color=white!55!black]
15        ([xshift=-2mm]frame.north west) -- ([xshift=2mm]frame.
16        north east)%
17        [rounded corners=1mm]-- ([xshift=1mm,yshift=-1mm]frame.
18        north east)%
19        -- (frame.south east) -- (frame.south west)%
20        -- ([xshift=-1mm,yshift=-1mm]frame.north west)%
21        [sharp corners]-- cycle;%
22      },interior engine=empty,%
23    },
24    enhanced jigsaw, before skip=2mm,after skip=2mm,%
25    fonttitle=\bfseries, #3,%
26    title={#2 \ifthenelse{\equal{#1}{}}{--~}{#1}}, %Aufgabe
27  }

```

Und die der Plain-Box:

```

1  \DeclareTColorBox{LILLYxBOXxAufgabexPlain}{0{} 0{} 0{}} {%
2    enforce breakable, enhanced jigsaw, before skip=2mm,after skip=
3    =2mm,%
4    colback=white,colframe=black!50,boxrule=0.2mm,fonttitle=
5    \bfseries,%
6    #3,title={#2 \ifthenelse{\equal{#1}{}}{--~}{#1}}%
7  }

```

## 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.



## 8.2.6 Weitere

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

# ANHANG

VERALTETE DOKUMENTE, ZUSÄTZLICHES, EASTER-EGGS, ...

9.1 Version 1.0.7

### 9.1.1 Installation in Linux

Da LILLY komplett auf einem Linux-Betriebssystem entwickelt wurde, gestaltet sich die Implementierung relativ einfach. Zuerst gilt es einen neuen Ordner zu erstellen:

```
1 mkdir -p "${HOME}/texmf/tex/latex/"
```

In diesen Ordner (wenn nicht sogar bereits existent) kann nun der gesamte `Lilly`-Ordner verschoben werden (oder mithilfe eines symbolischen Links verknüpft). Als letztes muss man nun noch `TEX` über das neue Verzeichnis informieren:

```
1 texhash "${HOME}/texmf"
```

Nun gilt es sich den anderen mitgelieferten Dateien zu widmen! Von besonderer Relevanz ist hierbei `lilly_compile.sh`, welches hier ausführlicher beschrieben wird (REMOVED: OLD). Grundslegend generiert es ein Makefile, das dann zum Kompilieren des Dokuments gedacht ist!

Mithilfe von folgendem Befehl wurde das Makefile für diese Dokumentation generiert:

```
1 ./lilly_compile.sh "Lilly-Dokumentation.doc.tex" \
2 -dir="Dokumentation/"
```

Hierbei wird das Makefile gemäß folgenden Regeln erzeugt:

- ◇ Es soll die tex-Datei: „Lilly-Dokumentation.doc.tex“ kompiliert werden.
- ◇ Das ganze soll (relativ zu `lilly_compile.sh`) im Verzeichnis `Dokumentation` stattfinden - hier wird ebenfalls das Makefile generiert.

### Bemerkung 9.1 – make

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

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

Mit diesem Makefile kann man nun das Dokument generieren lassen. Zu beachten sei hierbei, dass `make` - im Falle der Regel `all` - Regeln parallel ausführen wird!

Diese Dokumentation wurde mit folgendem Befehl erstellt:

```
1 make "BOXMODE=LIMERENCE"
```

Hierbei lässt sich ebenfalls erkennen wie sich noch mit dem Makefile einzelne Komponenten (wie das verwendete Boxdesign) ändern lassen!

VER 1.0.0

Es wird *nicht*  
auf die Semantik  
einzelner Befehle  
eingegangen!  
Copy&Paste ist  
doof, tippen! ;)

Dies sichert uns die Persistenz des Pakets im Falle einer Neuinstallation/Updates von L<sup>A</sup>T<sub>E</sub>X

VER 1.0.2

Es wird mit  
den Regeln  
default, all und  
clean generiert,  
selbstredend lässt  
sich dies erweitern

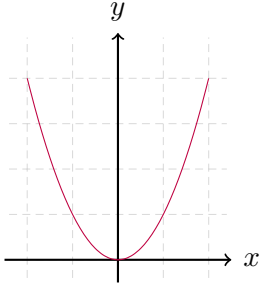
Die Anführungszeichen dienen hier und in anderen Codebeispielen lediglich zur Übersicht!

## 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) \&gt; 4       grid (2.4,4.9); 5   \draw[&gt;,thick] (-2.5,0) -- (2.5,0) 6     node[right]{\$x\$}; 7   \draw[&gt;,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}); 12 \end{tikzpicture} </pre>		<pre> 1 \begin{graph}[scale=0\&gt; 2   .6,domain=-2:2] 3   \plotline[purple]{\&gt; 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 du 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"
```

## Stichwortverzeichnis

<b>A</b>		<b>H</b>	
<code>\abs</code> (v1.0.9) . . . . .	9	<code>\Hcolor</code> (v1.0.9) . . . . .	27
<code>\ampelG</code> (v1.0.2) . . . . .	17	<b>I</b>	
<code>\arccot</code> (v1.0.8) . . . . .	10	<code>\i</code> (v1.0.1) . . . . .	10
<b>B</b>		<code>\Im</code> (v1.0.2) . . . . .	9
<code>\B</code> (v1.0.3) . . . . .	10	<code>\inf</code> (v1.0.6) . . . . .	9
<b>C</b>		<code>\isLanguageLoaded</code> (v2.0.0) . . . . .	31
<code>\ceil</code> (v2.0.0) . . . . .	13	<code>\isLanguageNameLoaded</code> (v2.0.0) . . . . .	31
<code>\cmdshowcase</code> (v2.0.0) . . . . .	32	<b>J</b>	
<code>\crossAT</code> (v1.0.1) . . . . .	11	<code>\join</code> (v2.0.0) . . . . .	11
<b>D</b>		<b>L</b>	
<code>\das</code> (v1.0.3) . . . . .	8	<code>\LILLYxColorxInject</code> (v1.0.1) . . . . .	27
<code>\det</code> (v1.0.3) . . . . .	9	<code>\LILLYxMathxMode</code> (v1.0.3) . . . . .	8
<code>\dif</code> (v2.0.0) . . . . .	10	<code>\LILLYxMODExEXTRA</code> (v1.0.4) . . . . .	19
<b>E</b>		<code>\LillyxStorexCurrentColorProfile</code> (v2.0.0)	27
<code>env@egraph</code> (v2.0.0) . . . . .	15	<code>\LILLYxwriteLst</code> (v1.0.8) . . . . .	32
<code>\enum</code> (v1.0.0) . . . . .	11	<code>\loopTo</code> (v1.0.3) . . . . .	21
<code>\epsilon</code> (v1.0.3) . . . . .	10	<code>\loopTop</code> (v1.0.3) . . . . .	21
<b>F</b>		<code>env@lstplain</code> (v1.0.9) . . . . .	32
<code>\folge</code> (v1.0.7) . . . . .	12	<b>M</b>	
<b>G</b>		<code>env@matrix</code> (v1.0.2) . . . . .	9
<code>\gdw</code> (v1.0.7) . . . . .	13	<b>N</b>	
<code>\getGraphics</code> (v2.0.0) . . . . .	22	<code>\N</code> (v1.0.0) . . . . .	10
<code>\getGraphicsPath</code> (v2.0.0) . . . . .	23	<code>\neuronSquare</code> (v1.0.5) . . . . .	22
<code>\getPrerendered</code> (v2.0.0) . . . . .	23	<code>env@nstabbing</code> (v1.0.2) . . . . .	11
<code>\gitRAW</code> (v1.0.0) . . . . .	33	<b>O</b>	
<code>env@graph</code> (v1.0.8) . . . . .	14	<code>\obda</code> (v1.0.8) . . . . .	13
<code>\graphdot</code> (v1.0.2) . . . . .	18	<code>\oragraphdot</code> (v1.0.2) . . . . .	18
<code>\graphPOI</code> (v1.0.4) . . . . .	19	<code>\overbar</code> (v1.0.3) . . . . .	8

<b>P</b>	<b>V</b>
<code>\PgetXY</code> (v2.0.0) ..... 19	<code>\val</code> (v1.0.8) ..... 10
<code>\plotline</code> (v1.0.8) ..... 13	<code>\VRule</code> (v1.0.4) ..... 12
<code>\plotseq</code> (v1.0.8) ..... 13	
<code>\POLYRAD</code> (v1.0.2) ..... 18	<b>W</b>
<b>R</b>	<code>env@wgraph</code> (v1.0.8) ..... 15
<code>\registerColors</code> (v2.0.0) ..... 25	<b>X</b>
<code>\reihe</code> (v1.0.7) ..... 12	<code>\x</code> (v1.0.2) ..... 13
<code>\rotateRPY</code> (v1.0.4) ..... 20	<code>\xa</code> (v1.0.1) ..... 11
<b>S</b>	<code>\xmark</code> (v2.0.0) ..... 13
<code>\sad</code> (v1.0.3) ..... 8	
<code>\sqrt</code> (v1.0.3) ..... 9	<b>Y</b>
<b>T</b>	<code>\ymark</code> (v2.0.0) ..... 14
<code>\trenner</code> (v1.0.0) ..... 12	