



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)

6. September 2019

Abstract

Oder auch Einleitung 🍓 für **VER 2.0.0**

Die \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

1 Einleitung	1
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	
2 Mathe	8
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
3 Grafiken	17
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
3.4 Weiterführende Symbole	24
3.4.1 Embleme	
4 Farben	27

4.1 Die normalen Farbprofile	27
4.1.1 Das Standardfarbprofil	
4.1.2 Das Druckprofil	
4.2 Farberweiterungen	30
4.3 Weitere Planungen	31
5 Listings	32
5.1 Die grundlegenden Eigenschaften	32
5.1.1 Grundlegendes Design	
5.1.2 Das MAIN-Paket	
5.1.3 Das MIPS-Paket	
5.1.4 Kontrolle der Sprachen	
5.2 Marker und weitere Befehle	41
5.2.1 Literates	
5.2.2 Marker	
5.3 Advanced Listings	42
5.4 Runtimes	43
6 Boxen	45
6.1 Grundlegendes	45
6.1.1 Eine kleine Einführung	
6.1.2 Der Box-Controller	
6.1.3 Die Boxmodi	
6.2 Info-Boxes	54
6.2.1 Wie es funktioniert	
7 Jake	58
7.1 Grundlegendes	58
7.1.1 Entwicklung	
7.1.2 Die Installation	
7.1.3 Lilly mit Jake installieren	
7.1.4 Jake im Überblick	
7.1.5 Entwicklerinformationen	
7.2 Gepard	63
7.2.1 Konfigurationsdateien	
7.2.2 Gepard Module im Allgemeinen	
7.2.3 Buildrules	
7.2.4 Expandables	
7.2.5 Hooks	
7.2.6 Name Maps	
8 Aussicht	71
8.1 Todos	71
8.1.1 Visuals	
8.1.2 Fehler	
8.1.3 Dateiaufteilung	
8.1.4 Road to CTAN	
8.1.5 Hoverover tooltips	
8.1.6 Weitere	

9 Anhang	72
9.1 Version VER 1.0.7	72
9.1.1 Installation in Linux	
9.1.2 Spezifikation: Plots	
9.2 Version VER 1.0.9	74
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 gearbeitet.

VER 2.0.0

Bemerkung 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 < 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. Bei der Installation gilt es die Angabe einer [Nutzerkonfiguration](#) zu beachten. 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 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[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 nötig nur `Dokumentation` anstelle von `Typ=Dokumentation` zu schreiben, da die explizite Zuweisung versucht, auf die entsprechende Datei zu referenzieren. 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 1.1 – 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}
```

Lemma 1.1

Hallo Welt

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

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 % \def\LILLYxBOXx<FirstLetterUp-Name>xEnable{FALSE}
3 \def\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

1.1	Titel	4
6.1	Titel	47
6.2	📌 Titel	47

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 mich⁵)

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          = @[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{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 3 – Standalone-Math

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

```
\usepackage{LILLYxMATH}
```

auch ohne das Verwenden der restlichen LILLY-Welt benutzen lässt.

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>	$\overline{a_1 a_2}$
<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 `\vcntcolon` 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>\abs{\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:

<pre> 1 \$\begin{pmatrix}cc c 2 1 & 2 & 3 \\ 3 4 & 5 & 6 4 \end{pmatrix}\$ </pre>	$\left(\begin{array}{cc c}1 & 2 & 3 \\ 4 & 5 & 6\end{array}\right)$
---	---

◇ `\val`, `\sch`, `\dom`, `\grad`

v1.0.8

Auch hier handelt es sich um weitere Mathe-Operatoren, die selbstredend implementiert werden:

◇ `\val` (val)◇ `\dom` (dom)◇ `\sch` (sch)◇ `\grad` (Grad)

◇ `\arccot`

v1.0.8

Da der ach so wichtige Arkuskotangens erstaunlicherweise nicht standardmäßig dabei ist, hier: `\arccot` (arccot).

◇ `\dif`, `\dint[Variable=<x>]`

v2.0.0

Auch hierbei handelt es sich wieder um stupide Abbildungen im Operator-Style für Integration und Differenzierung

◇ `\dif` (d)◇ `\dint` ($\frac{d}{dx}$)

2.1.2 Symbole

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

◇ `\N`, `\Z`, `\Q`, `\R`, `\C`

v1.0.0

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.

◇ `\i`

v1.0.1

Die komplexe Einheit i wird mit `\i` zur Verfügung gestellt.

◇ `\epsilon`, `\phi`

v1.0.3

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

◇ `\B`, `\X`, `\K`, `\P`, `\F`, `\O`

v1.0.3

Zudem wird zum Beispiel die Menge der Binärzahlen über `\B` (\mathbb{B}), die chromatische

Zahl über $\backslash\mathbb{X}$ (χ) und der generelle Körper mit $\backslash\mathbb{K}$ (\mathbb{K}) zur Verfügung gestellt. Für die Potenzmenge liefert LILLY $\backslash\mathcal{P}$ (\mathcal{P}), für die Menge der Funktionen $\backslash\mathcal{F}$ (\mathcal{F}) und für die Groß-O-Notation $\backslash\mathcal{O}$ (\mathcal{O}).

◇ $\backslash\join$, $\backslash\leftouterjoin$, $\backslash\rightouterjoin$, $\backslash\fullouterjoin$

v2.0.0

Da auch die Relationenalgebra Teil der Mathematik ist, hier die entsprechenden Symbole für die Joins:

- | | |
|--|--|
| ◇ $\backslash\join$ (\bowtie) | ◇ $\backslash\rightouterjoin$ ($\bowtie\! \! \! \bowtie$) |
| ◇ $\backslash\leftouterjoin$ ($\bowtie\! \! \! \bowtie$) | ◇ $\backslash\fullouterjoin$ ($\bowtie\! \! \! \bowtie\! \! \! \bowtie$) |

Bemerkung 4 – Weitere Symbole

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

2.1.3 Kompatibilität

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

Hier werden einige Befehle eingerichtet, die entweder noch nicht zugeordnet wurden `VER 2.0.0` oder während der Vorlesung (im Überlebenskampf :P) ins `eagleStudiPackage` eingebaut worden sind.

◇ $\backslash\enum\{items}$, $\backslash\liste\{items}$

v1.0.0

Hier befinden sich die für *Lineare Algebra* kreierten: $\backslash\enum\{items}$ (`enumerate` mit $\backslash\narrowitems$) und $\backslash\liste\{items}$ (`enumerate` mit römischen Zahlen und $\backslash\narrowitems$).

◇ $\backslash\mathbf{x}_a$, $\backslash\mathbf{x}_b$, $\backslash\mathbf{x}_c$

v1.0.1

Weiter existieren die Befehle $\backslash\mathbf{x}_a$ ($\overline{x_1}$), $\backslash\mathbf{x}_b$ ($\overline{x_2}$), $\backslash\mathbf{x}_c$ ($\overline{x_3}$), welche einen etwas größeren Abstand für eine bessere Lesbarkeit einfügen.

◇ $\backslash\crossAT\{(PosX,PosY)\}$, $\backslash\circAT\{(PosX,PosY)\}$, $\backslash\block\{(PosX,PosY)\}\{text\}$

v1.0.1

Für TikZ gibt es noch die Befehle $\backslash\crossAT\{(PosX,PosY)\}$ ($\mathbf{x}^{(a)}$) und analog $\backslash\circAT\{(PosX,PosY)\}$ ($\mathbf{O}^{(b)}$), sowie $\backslash\block\{(PosX,PosY)\}\{text\}$ ($\boxed{42}^{(c)}$). Hier fragt man sich nun vielleicht, warum diese nicht in einem entsprechenden TikZ-Paket sind. Im Rahmen der mit `VER 2.0.0` eingeführten Modularisierung hat sich diese Verteilung als günstig erwiesen.

◇ $\mathbf{env@nstabbing}$, $\mathbf{env@centered}$, $\mathbf{env@sqcases}$

WAR Veraltet

v1.0.2

Weiter werden drei (mittlerweile obsolete) Umgebungen definiert:

- ^(a) $\backslash\tikz\{\backslash\crossAT\{(0,0)\};\}$ – Zum Erhalt der Textzeile vertikal um $-0.35\backslash\baselineskip$ verschoben.
^(b) $\backslash\tikz\{\backslash\circAT\{(0,0)\};\}$
^(c) $\backslash\tikz\{\backslash\block\{(0,0)\}\{42\};\}$ – Wieder vertikal um $-0.2\backslash\baselineskip$ verschoben.

- ◇ `env@nstabbing`: tabbing-Umgebung, ohne Abstände
- ◇ `env@centered`: center-Umgebung, ohne Abstände
- ◇ `env@sqcases`: Ähnelt `cases` - nur mit `'`'.

◇ `\VRule{width}`

v1.0.4

Zudem definiert sich noch für Tabellen der Befehl `\VRule{width}`, welcher eine Spalte variabler Größe für Tabellen zur Verfügung stellt. Eine exemplarische Verwendung 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>	
--	--

◇ `\trenner` WAR Veraltet

v1.0.0

Fügt einen großen senkrechten Strich ein: `\trenner` (`|`).

2.1.4 Shortcuts

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

Hier befinden sich einige abkürzende Befehle, die primär das Schreiben beschleunigen sollen. Sie werden auf Bedarf stetig erweitert.

◇ `\folge[Folgenglied=<a>]`

v1.0.7

Setzt eine Folge, welche mit dem Index n arbeitet: `\folge` $((a_n)_{n \in \mathbb{N}})$

◇ `\reihe[Folgenglied=<a_k>][Start=<0>]`

v1.0.7

Setzt eine Reihe über die Glieder *Folgenglied* an *Start*: `\reihe` $(\sum_{k=0}^{\infty} a_k)$

◇ `\obda`, `\Obda`

v1.0.8

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

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

v1.0.7

Setzt verschiedene mathematische Ausdrücke:

- ◇ `\gdw` (\Leftrightarrow)
- ◇ `\sumn` ($\sum_{n=0}^{\infty}$)
- ◇ `\sumk` ($\sum_{k=0}^{\infty}$)
- ◇ `\limn` ($\lim_{n \rightarrow \infty}$)
- ◇ `\limk` ($\lim_{k \rightarrow \infty}$)

◇ `\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\text{Ausdruck}\right\rfloor` beziehungsweise `\lceil` & `\rceil` entsprechend:

- ◇ `\ceil` ($\lceil \frac{a}{b} \rceil$)
- ◇ `\floor` ($\lfloor \frac{a}{b} \rfloor$)

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`^{v2.0.0} lässt sich die Funktion entsprechend verschieben.

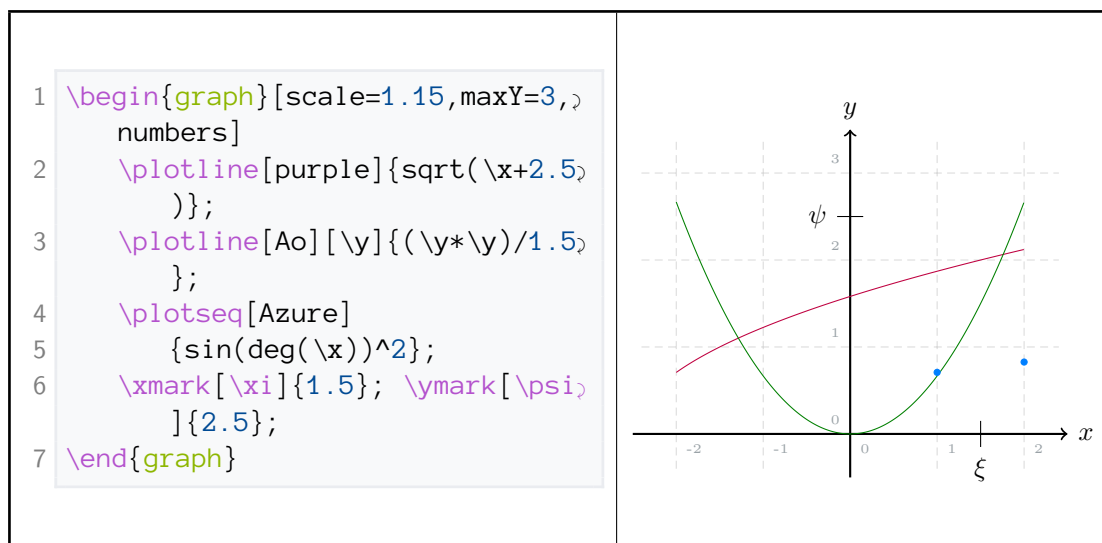
◇ `\plotseq[Farbe=<Ao>][Variable=<x>]{Term}[Obergrenze=<maxX>][Untergrenze=<1>][Dicke=<1pt>]` v1.0.8
 Zeichnet in eine **Graph-Umgebung** eine Folge zwischen *Unter-* und *Obergrenze* mit Punkten der Größe *Dicke* (siehe Umgebung für Beispiel). Existiert auch außerhalb von `env@graph`, ist aber hier 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 *Konfiguration* gibt es die folgenden Parameter:

Bezeichner	Typ	Standard	Beschreibung
scale	Zahl	1	Skalierungsfaktor
xscale	Zahl	1	x -Skalierungsfaktor ^{v2.0.0}
yscale	Zahl	1	y -Skalierungsfaktor ^{v2.0.0}
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	Zahl	0	Nummernstart y
numbersize	Zahl	5	Schriftgröße Nummerierung
labelsize	Zahl	10	Schriftgröße Texte

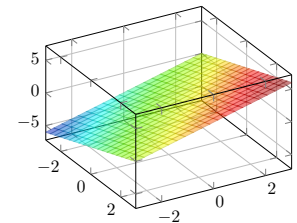
2.3 3D-Plots

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}

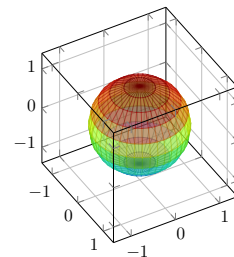
```



```

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 5 – Standalone-Graphics**

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

```
\usepackage{LILLYxGRAPHICS}
```

auch ohne das Verwenden der restlichen LILLY-Welt benutzen lässt.

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_AMPEN`. 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 6 – 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 Farbbargument 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`, `\blagraphdot`, `\nographdot`, `\margraphdot` v1.0.2

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>\begin{tikzternal}[scale=0.75, every node/.style={transform shape}] \graphPOI{(0,0)}{purple}{1999 n.Chr.} {Florian Sihler} {Florian Sihler ist der Autor dieses Dokuments.} {Data/2003.jpg} {https://github.com/EagleoutIce/Quickblit} {Deutschland}; \end{tikzternal}</pre>	<div style="display: flex; justify-content: space-between; align-items: flex-start;"> <div style="text-align: left;"> <p>— 1999 n.Chr.</p> <p>Florian Sihler: 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:

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

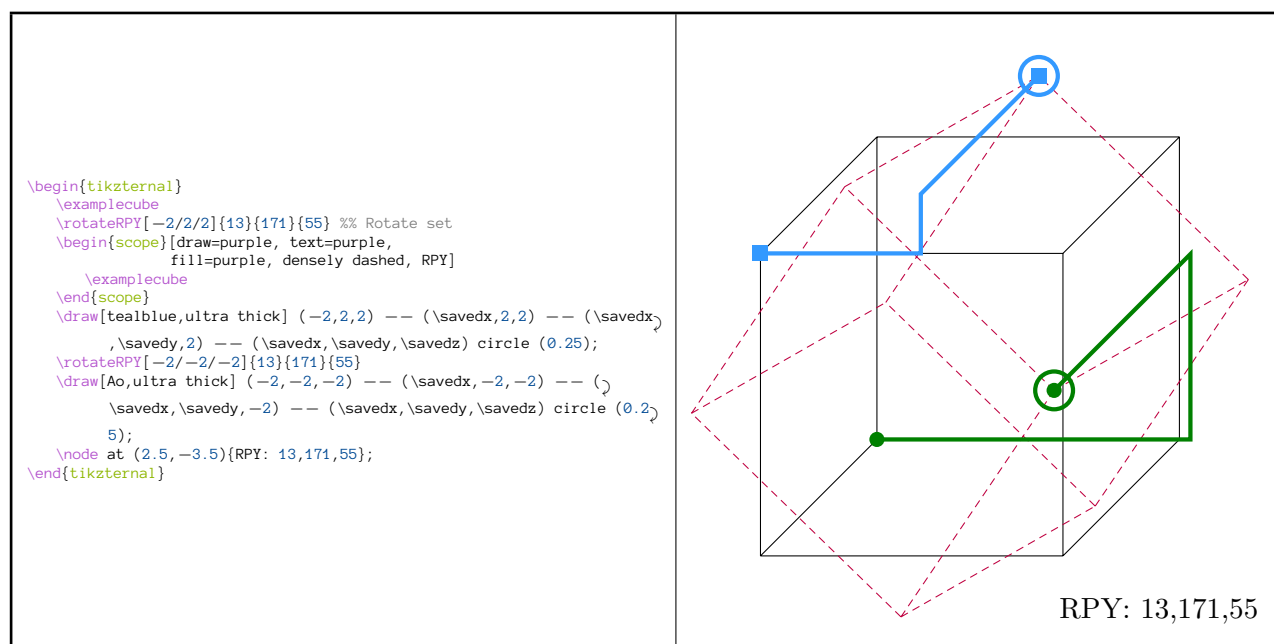
3.2.2 Rotation

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^(a) 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

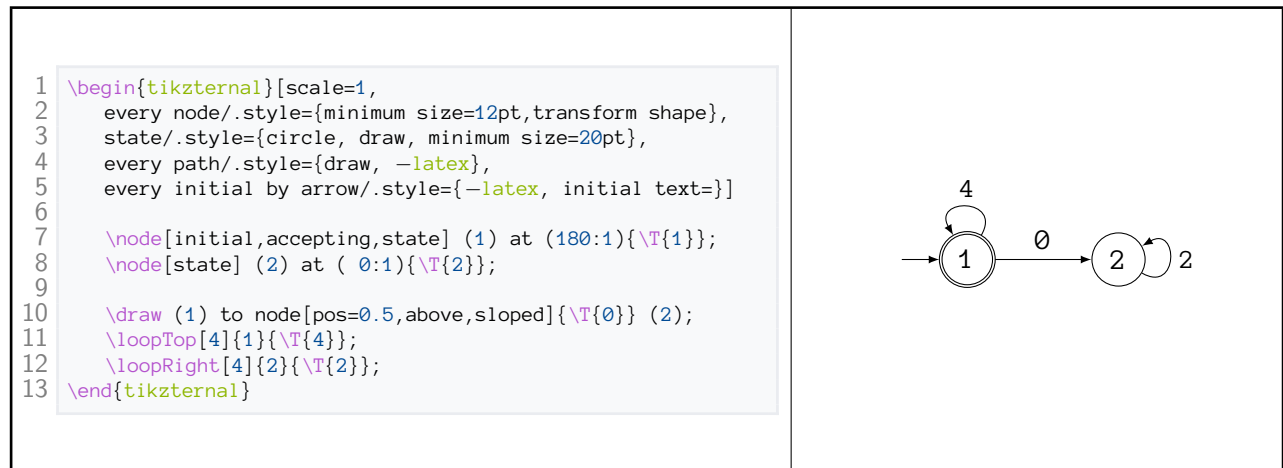
^(a)<https://tex.stackexchange.com/questions/67573/tikz-shift-and-rotate-in-3d>

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

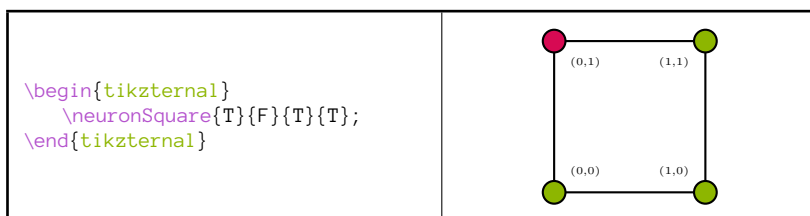
LIB LILLYxGRAPHICS geladen.

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

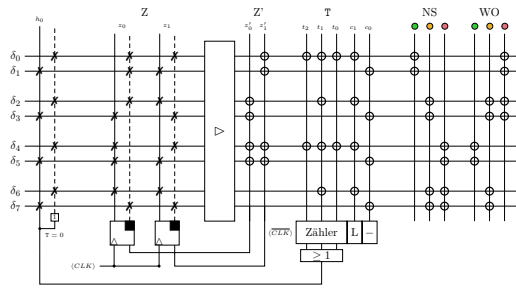
Diese Definitionen befinden sich in der Datei: `source/Graphics/LILLYxGRAPHICSxPROVIDER.sty`. Sie werden mit VER 2.0.0 automatisch mit dem Einbinden von LIB LILLYxGRAPHICS geladen.

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

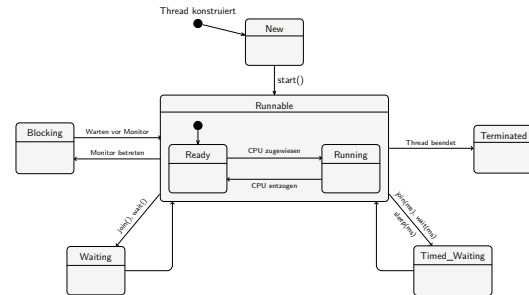
◇ `\getGraphics[width=(\linewidth)]{path}[height]`

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:

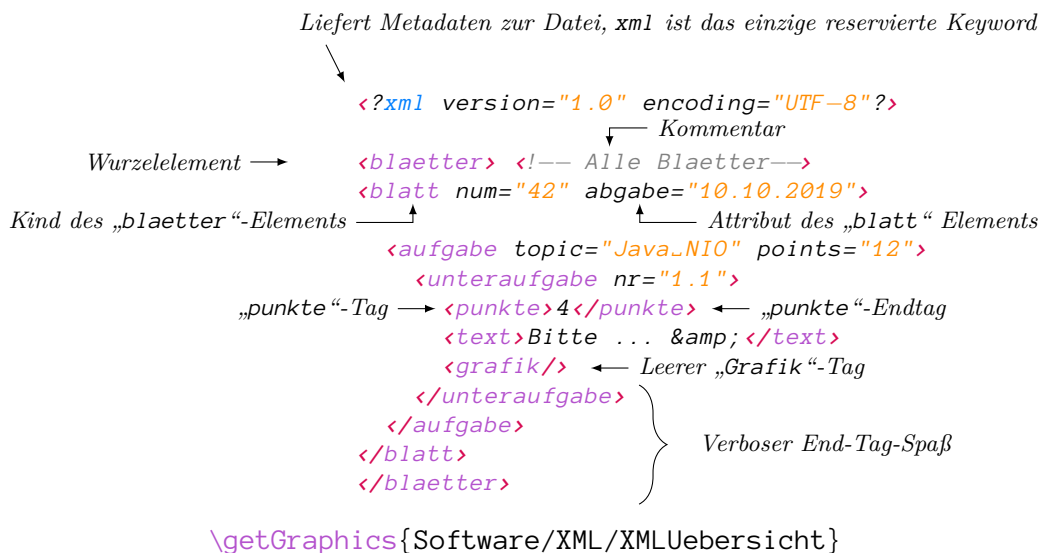


\getGraphics{Rechner/PLAAmpel}



\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:



◇ \getGraphicsPath{path}

v2.0.0

Liefert den absoluten Pfad zu einer Grafik. Beispiel:

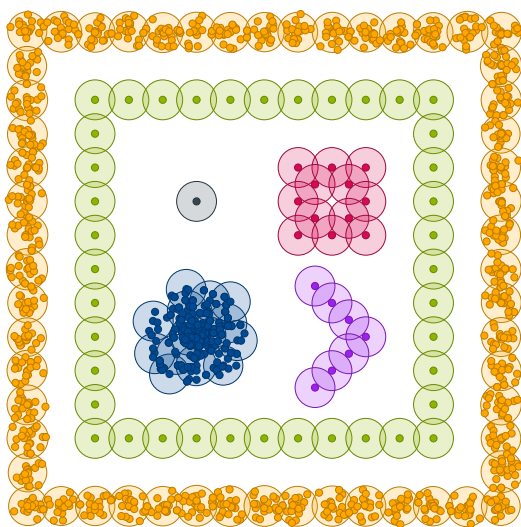
```
\getGraphicsPath{Software/XML/XMLUebersicht}
```

Liefert: source/Data/Graphics/Software/XML/XMLUebersicht.

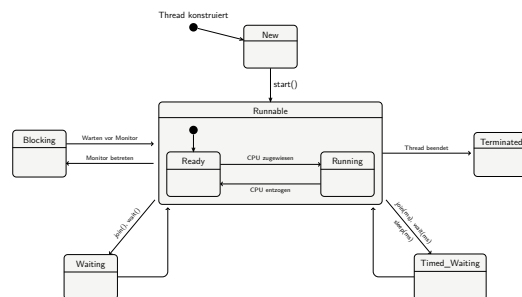
◇ \getPrerendered[width=(\linewidth)][path][height]

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!

3.4 Weiterführende Symbole

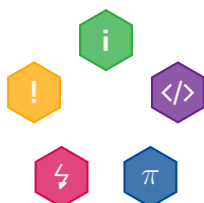
3.4.1 Embleme

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

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

- ◇ `\infoEmblem` v2.0.0
- ◇ `\warningEmblem` v2.0.0
- ◇ `\errorEmblem` v2.0.0
- ◇ `\mathEmblem` v2.0.0
- ◇ `\codeEmblem` v2.0.0

Hierbei handelt es sich um Shortcuts um einige Embleme direkt zu Setzen:



Hierbei bedienen sich die Befehle der Emblem-Definition `\DefaultBaseEmblem`.

- ◇ `\NewEmblem[Emblem-Keys][Tikz-Args]{name}` v2.0.0

Definiert ein neues Emblem, wobei folgende Emblem-Keys zur Verfügung stehen, diese werden persistiert:

Bezeichner	Typ	Standard	Beschreibung
radius	<i>Length</i>	0.369cm	Radius des Symbols
shape	<i>Enum</i> ^(b)	shape/hexagon	Form des Hintergrunds
bgcolor	<i>Farbe</i>	DebianRed	Hintergrundsfarbe
bordercolor	<i>Farbe</i>	DebianRed	Rahmenfarbe
fgcolor	<i>Farbe</i>	MudWhite	Textfarbe
font	<i>Code</i>	<code>\bfseries\large\sffamily</code>	Schrift

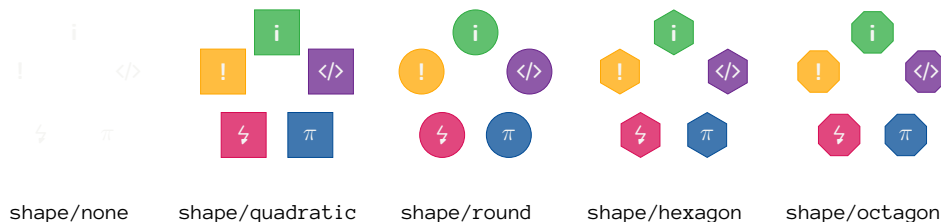
So lassen sich relativ einfach GrundEmbleme definieren:

```

1 \NewEmblem[shape/none]{NoneEmblem}
2 \NewEmblem[shape/quadratic]{QuadraticEmblem}
3 \NewEmblem[shape/round]{RoundEmblem}
4 \NewEmblem[shape/hexagon]{HexagonEmblem}
5 \NewEmblem[shape/octagon]{OctagonEmblem}

```

So ist es zum Beispiel möglich durch die jeweilige Form das aussehne der mitgelieferten Embleme zu modifizieren:



Das Erzeugen eines neuen Emblems mithilfe von `\NewEmblem` erzeugt einen neuen Befehl, entsprechend des Namens des Emblems. Der Befehl besitzt jeweils die folgende Signatur:

◇ `\<name>[Tikz-Keys]{text}`

v2.0.0

So liefert zum Beispiel: `\OctagonEmblem{Hu}`: .

Oder: `\OctagonEmblem{\tiny stop}` .

Die Shortcuts von oben, wurden hierbei wie folgt definiert:

```

1 \gdef\infoEmblem{\, \DefaultBaseEmblem[draw=Leaf, fill=Leaf!75]{i}\,}
2 \gdef\warningEmblem{\, \DefaultBaseEmblem[draw=ChromeYellow, fill=ChromeYellow!75]{!}\,}
3 \gdef\errorEmblem{\, \DefaultBaseEmblem[draw=DebianRed, fill=DebianRed!75]{\wasysymLightning}\,}

```

^(b) Allowed: none, quadratic, round, hexagon, octagon

```

4 \gdef\mathEmblem{\, \DefaultBaseEmblem[draw=DarkMidnightBlue,fill=
  DarkMidnightBlue!75]{$\mathbf{\pi}$}\,}
5 \gdef\codeEmblem{\, \DefaultBaseEmblem[draw=DarkOrchid,fill=DarkOrchid
  !75]{\faCode}\,}

```

◇ `\textEmblem{Emblem}`

v2.0.0

Setzt ein Emblem für den Fließtext: anstelle von . Die Argumentklammern können Vernachlässigt werden, das bedeutet es genügt das Schreiben von `\textEmblem\codeEmblem`.

◇ `\btextEmblem{Emblem}`

v2.0.0

Funktioniert identisch, setzt allerdings ein Emblem, welches die komplette Zeilenhöhe ausfüllt: anstelle von .

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 7 – Standalone Color

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

```
\usepackage{LILLYxCOLOR}
```

auch ohne das Verwenden der restlichen LILLY-Welt benutzen lässt.

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 8 – 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>)
● Übungsaufgabe (<i>Veronica</i>)	● Übungsaufgabe (<i>Charcoal</i>)
● Zusatzübung (<i>Veronica</i>)	● Zusatzübung (<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 standardmäßig 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:

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

Bemerkung 9 – Standalone Listings

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

```
\usepackage{LILLYxLISTINGS}
```

auch ohne das Verwenden der restlichen LILLY-Welt benutzen lässt.

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: `\LILLYxPATHxLISTINGS/LILLYxLISTINGS`. Sie werden mit VER 2.0.0 automatisch mit dem Einbinden von LIB LILLYxLISTINGS geladen.

Hier wird weiter von der Bibliothek LIB LILLYxLISTINGSxLANGUAGExCONTROL gebrauch gemacht, die sich mittels `\RegisterLanguage` um die Konstruktion der im Folgenden vermerkten Möglichkeiten kümmert!

Bemerkung 10 – 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. minted wird nicht verwendet um die Portabilität zu gewährleisten. Allerdings erlaubt `\RegisterLanguage` das Verwenden von minted.

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:

```
\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>[Listing-Options]{Code}`

v1.0.9

Setzt den Code mit grauem Hintergrund. Zeilenumbrüche werden hier zwar durchgeführt, allerdings in der Regel nicht optimal gesetzt. Beispiel:

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

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

◇ `\b<Sprache>[Listing-Options]{Code}`

v1.0.9

Setzt den Code farbig auf dem vorhandenen Hintergrund. Beispiel:

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

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

◇ `\p<Sprache>[Listing-Options]{Code}`

v2.0.0

Setzt den Code im Präsentationsstil. Beispiel:

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

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

◇ `\i<Sprache>[Listing-Options]{Code}`

v1.0.9

Lädt und setzt den Programmcode aus der entsprechenden Datei. Beispiel:

```
\ilatex[firstline=5,lastline=10]{Data/Listings.doc.tex}
```

Liefert:

```
1 \elable{chp:LISTINGS}\hypertarget{LILLYxLISTINGS}Zum Setzen von
   Programmtexten innerhalb von Latexdokumenten stellt dieses Paket
   eine große Ansammlung verschiedener Sprachen und Dialekten zur
   Verfügung. Es befindet sich hier:
2 \begin{center}
3   \blankcmd{LILLYxPATHxLISTINGS} = \T{\LILLYxPATHxLISTINGS}
4 \end{center}
5
6 \begin{bemerkung}[Standalone Listings]
```

◇ `env@<Sprache>[Listing-Options]`

v1.0.9

Erlaubt das Setzen eines Textblocks in der jeweiligen Sprache, es ist die Kurzform von:

```
1 \begin{lstlisting}[style=\pgfkeysvalueof{/lilyxLISTINGS/globals/}
   listing style],language=<Sprache>]
2   Hier kann Code in der jeweiligen Sprache stehen.
3 \end{lstlisting}
```

Dieses Beispiel wurde zum Beispiel durch die Sprache `latex` gesetzt. Beispiel:

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

Ergibt:

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

◇ `env@<Sprache>*[Listing-Options]`

v1.0.9

Entfernt die Zeilennummern eines sonst standardmäßigen Listings:

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

Ergibt:

```
public class SuperKlasse {
    public static void main(String[] args) {
        System.out.println("Hallo Welt");
    }
}
```

◇ `env@plain<Sprache>[Listing-Options]`

v2.0.0

Setzt ein Listing ohne irgendwelche zusätzlichen graphischen Hervorhebungen, außer sie werden durch die Optionen angegeben. Beispiel:

```

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

```

Ergibt:

```

public class SuperKlasse {
    public static void main(String[] args) {
        System.out.println("Hallo Welt");
    }
}

```

◇ `env@s<Sprache>`

v2.0.0

Setzt ein Listing im Showcase-Design. Beispiel:

```

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

```

Ergibt:

```

1 public class SuperKlasse {
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

```

◇ `\lstshowcmd{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. \lstshowcmd[language=lJava]{\x}
5     }
6   \end{ditemize}
7 \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 11 – 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.

◇ `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");
}
```

Die allgemeine TypeWriter-Schriftart wird übrigens mithilfe von `\LILLYxlstTypeWriter` auf AnonymousPro gesetzt (*Sie wird auch hier für die Dokumentation verwendet*).

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.

Neben den geladenen Sprachen, liefert dieses Paket die Stildefinitionen die bereits in kleinen Teilen auch über PGF-Konfiguriert werden können. Dies wird aber wohl erst in zukünftigen Versionen sinnvoll konfigurierbar sein.

◇ `\lstcomment{text}`, `\lststring{text}`, `\lstnumber{text}`

v2.0.0

Setzt den Text so wie der Main-Stil den Code als Kommentar, String oder Zahl setzen würde. So kann auch durch `!* ... *` gesetzter Code korrekt formatiert werden:

```
1 \begin{java}
2 5 + 3 ergibt: !*\pgfmthparse{5+3}\pgfmthresult*!
3 5 + 3 ergibt: !*\lstnumber{\pgfmthparse{5+3}\pgfmthresult}*!
4 \isLanguageNameLoaded{java} // :yields: !*\isLanguageNameLoaded{java}
   *!
5 \isLanguageNameLoaded{java} // :yields: !*\lstcomment{
   \isLanguageNameLoaded{java}}*!
6 "Im !*\LILLYxDOCUMENTxSUBNAME*! :D"
7 "Im !*\lststring{\LILLYxDOCUMENTxSUBNAME}*! :D"
8 \end{java}
```

Ergibt^(a):

```
1 5 + 3 ergibt: 8.0
2 5 + 3 ergibt: 8.0
3 \isLanguageNameLoaded{java} // → TRUE
4 \isLanguageNameLoaded{java} // → TRUE
5 "Im_Data/Listings.doc :D"
6 "Im_Data/Listings.doc :D"
```

◇ `\lstkwone{text}`, `\lstkwtwo{text}`, ..., `\lstkwsix{text}`

v2.0.0

Setzt den Text wie das entsprechende Keyword-Level:

```
1 \begin{java}
2 !* Hallo *! !* \lstkwone{Hallo} *! !* \lstkwtwo{Hallo} *!
3 !* \lstkwtthree{Hallo} *! !* \lstkwfour{Hallo}* !
```

^(a) Hier werden die Befehle nicht richtig markiert, da zum veranschaulichen von `\lststring` eine Sprache nötig war, die Zeichenketten als Datentyp besitzt.

```

4 !* \lstkwfive{Hallo} *! !* \lstkwsix{Hallo} *!
5 \end{java}

```

Ergibt:

```

1 Hallo  Hallo  Hallo
2 Hallo  Hallo
3 Hallo  Hallo

```

Bemerkung 12 – 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

MAIN lädt noch das Paket MIPS, auf welches nun noch etwas weiter eingegangen wird...

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/). 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 13 – MIPS

Das gesamte Mipspaket ist seit VER 1.0.8 überholt und bedarf einiger Aufarbeitung, dennoch tut es seinen Dienst für die bisher existenten MIPS-Codes. Weitere Besonderheiten wie zum Beispiel Literates nebst der anfänglich implementierten stehen *nicht* zur Verfügung...

5.1.4 Kontrolle der Sprachen

Mit VER 2.0.0 läuft die Registrierung einer Sprache über das Sub-Paket LIB LILLYxLISTINGSxLANGUAGExCONTROL ab. Dieses definiert eine Menge an Befehlen, im Kern ist allerdings nur folgender von Relevanz:

◇ `\RegisterLanguage{Sprache}{lst-language}` v2.0.0
`[sig-list=<RegisteredLanguages>]`
`[name-list=<RegisteredLanguageNames>]`

Registriert eine Sprache wie java mit der entsprechenden Listing-Sprache `lst-language` wie `lJava`. Die Signatur der Sprache wird in die Liste `sig-list`, der ledigliche Name (wie `java`) in die Liste `name-list` eingetragen. Konstruiert werden die **weiter oben** beschriebenen Befehle für die Sprache. Erschaffen wir uns einmal die Sprache `rubberduck`:

```
1 \lstdefinlanguage{1Rubberduck}{
2   comment=[1]{\#},
3   morekeywords = {Quack, new},
4   morekeywords = [2]{Duck}
5 }
```

Bisher haben wir damit noch keine Sprache geladen oder irgendetwas Lilly-Kompatibles erzeugt:

```
\isLanguageNameLoaded{rubberduck} % → FALSE
```

Dies ändert sich durch folgenden Befehl:

```
\RegisterLanguage{rubberduck}{1Rubberduck}
```

Nun gilt die Sprache als geladen:

```
\isLanguageNameLoaded{rubberduck} % → TRUE
```

Und wir können sie im Code auch verwenden:

```
1 \begin{rubberduck}
2 Duck jens = new Duck(); # Eine neue Ente
3 Quack jens ::{
4   Quack Quack, Quack Quack
5   Quack Quack. # Entisch, es ist so simpel
6 }
7 \end{rubberduck}
```

Ergibt:

```
1 Duck jens = new Duck(); # Eine neue Ente
2 Quack jens ::{
3     Quack Quack, Quack Quack
4     Quack Quack. # Entisch, es ist so simpel
5 }
```

Analog existieren auch die inline-Befehle:

```
\prubberduck{Duck primus = new Duck();}
```

Ergibt: `Duck primus = new Duck();`

Wie die einzelnen Umgebungen heißen und wie sie dargestellt werden sollen lässt sich relativ frei konfigurieren. Für die durch `\lillylstset`-modifizierbaren Schlüssel (wie Präfix und Suffix des Befehls) steht die Dokumentation noch aus!

◇ `\LillyNewLstEnvironCore{Name}{Key}{In-Extra}{Out-Extra}{Language}` v2.0.0

Dieser Befehl sollte nicht manuell aufgerufen werden, er wird aufgerufen und kann somit vom Nutzer modifiziert/überschrieben werden um die Eigenschaften der durch `\RegisterLanguage`-generierten Umgebungen zu modifizieren. Diese Befehl kümmert sich um die Standartumgebung wie `\begin{latex}`. Er erhält die entsprechenden Informationen über die jeweiligen Argumente. Die Standarddefinition dieses Befehls lautet ganz einfach:

```
1 \def\LillyNewLstEnvironCore#1#2#3#4#5{%
2     \lstnewenvironment{#1}[1][\{#3\lstset{##1}\}{#4}
3 }
```

◇ `\LillyNewLstEnvironPlain{Name}{Key}{In-Extra}{Out-Extra}{Language}` v2.0.0

Dieser Befehl sollte nicht manuell aufgerufen werden, er wird aufgerufen und kann somit vom Nutzer modifiziert/überschrieben werden um die Eigenschaften der durch `\RegisterLanguage`-generierten Umgebungen zu modifizieren. Diese Befehl kümmert sich um die Plain-Umgebung wie `\begin{plainlatex}`. Die Standarddefinition dieses Befehls lautet:

```
1 \def\LillyNewLstEnvironPlain#1#2#3#4#5{%
2     \lstnewenvironment{#1}[1][\{#3\lstset{xleftmargin=0pt,
3         xrightmargin=0pt,%
4         numbers=none,numbersep=0pt,frame=none,%
5         rulecolor={},backgroundcolor={},##1}\}{#4}
```

◇ `\LillyNewLstEnvironPresent{Name}{Key}{In-Extra}{Out-Extra}{Language}` v2.0.0

Dieser Befehl sollte nicht manuell aufgerufen werden, er wird aufgerufen und kann somit vom Nutzer modifiziert/überschrieben werden um die Eigenschaften der durch `\RegisterLanguage`-generierten Umgebungen zu modifizieren. Diese Befehl kümmert sich um die Presentation-Umgebung wie `\begin{slatex}`. Die Standarddefinition dieses Befehls lautet:

```

1 \def\LillyNewLstEnvironPresent#1#2#3#4#5{%
2   \expandafter\xdef\csname#1\endcsname{\noexpand\leavevmode}
3   \noexpand\presentlst{#5}}
4   \expandafter\xdef\csname end#1\endcsname{\noexpand\endpresentlst}
5 }

```

◇ `env@presentlst[lst-args]{language},`
`env@plainlst[lst-args]{language},`
`env@defaultlst[lst-args]{language}`

v2.0.0

Liefert die Listings-Umgebungen jeweils als `tcblisting`. Beispiel:

```

1 \begin{defaultlst}{lJava}
2 System.out.println("Hallo Welt");
3 \end{defaultlst}
4 \begin{plainlst}{lJava}
5 System.out.println("Hallo Welt");
6 \end{plainlst}
7 \begin{presentlst}{lJava}
8 System.out.println("Hallo Welt");
9 \end{presentlst}

```

Ergibt:

```
1 System.out.println("HalloWelt");
```

sowie:

```
System.out.println("HalloWelt");
```

und:

```
1 | System.out.println("HalloWelt");
```

Letzere Box wird auch für den generierten Befehl verwendet. Die Inline-Befehle verwenden jeweils `\LILLYxLSTINLINE`, `\LILLYxLSTBLANKINLINE`, `\LILLYxLSTINPL` und `\LILLYxLSTINLINExADVANCED`.

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:

<code>:bs: „\“</code>	<code>:ws: „ “</code>	<code>:float: „f“</code>	<code>:bcmd: „\“</code>
<code>:bmath: „\$“</code>	<code>:cdots: „...“</code>	<code>:exp: „e“</code>	<code>:star: „*“</code>
<code>:emath: „\$“</code>	<code>:cdot: „·“</code>	<code>:yields: „→“</code>	
<code>:dollar: „\$“</code>	<code>:ldots: „...“</code>	<code>:lan: „{“</code>	
<code>:space: „ “</code>	<code>:c: „“</code>	<code>:ran: „}“</code>	

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 \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}

```

Ergibt:

```

1 import java.util.ArrayList;
2
3 public class Example {
4     public static void main(String[] args) {
5         System.out.println("Hallo Welt");
6         if(args==null)
7             System.out.println("wau");
8     }
9 }

```

5.3 Advanced Listings

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

◇ `env@presentlst[tcbaddons]{language}` v2.0.0

Setzt Quellcode in einem modernen Design:

```

1 \begin{presentlst}{lJava}
2 public static void main(String[] args) {
3     System.out.println("Hallo Welt");

```

```

4 }
5 \end{presentlst}

```

Ergibt:

```

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

```

◇ `\p<lang>{Code}`

v2.0.0

Setzt Analog zu `\c<lang>` den Code in einer Zeile im entsprechend Design. Hier allerdings ebenfalls das neue, modernere Design:

```

1 \pcpp{int main(int argc, char** argv)}

```

Ergibt: `int main(int argc, char** argv)`.

5.4 Runtimes

Diese Definitionen befinden sich in der Datei: `\LILLYxPATHxLISTINGS/LILLYxRUNTIMES`. Sie werden mit `VER 2.0.0` automatisch mit dem Einbinden von `LIB LILLYxLISTINGSxADVANCED` geladen. Weiter existiert es als eigenständiges Paket `LIB LILLYxRUNTIMES`.

Runtimes bieten die Möglichkeit Code einer Programmiersprache in Latex ausführen zu lassen und das Ergebnis ebenfalls im Latexdokument zu setzen. Hierfür wird eine bereits aufgesetzte Umgebung für die jeweilige Sprache benötigt, LILLY greift also auf einen bestehenden Compiler/Interpreter zurück. Alle mitgelieferten Runtimes befinden sich in der Liste `RegisteredRuntimes` und liefern:

◇ `\r<Runtime>[Mid-Text=<\,:>]{Code}`

v2.0.0

Führt den übergebenen Code in der jeweiligen Runtime aus und liefert das Ergebnis. So zum Beispiel mit `\rbash{ls . | tail -2}`: `ls . | tail -2`:

```

Lilly-Dokumentation.doc.upa
README.md

```

`\rbash[liefert:]{ls . | tail -4}`: `ls . | sort | tail -4` liefert:

```

Lilly-Dokumentation.doc.TOP
Lilly-Dokumentation.doc.txt
Lilly-Dokumentation.doc.upa
README.md

```

◇ `\isRuntimeLoaded{runtimeName}`

v2.0.0

Testet analog zu `\isLanguageNameLoaded` ob eine entsprechende Runtime geladen ist:

```

1 \isRuntimeLoaded{bash} % → TRUE
2 \isRuntimeLoaded{waffel} % → FALSE

```

Bemerkung 14 – Was es noch so gibt

Die Runtimes liefern, bisher noch nicht normiert, auch noch Befehle wie: `\preview-BashFile`, die eine bestehende Datei ausführen und das Ergebnis ausgeben. An einer Normierung und Erweiterung wird gearbeitet.

6

BOXEN

BOXES IN BOXES IN BOXES IN BOXES...

VER 1.0.0

Boxen aller Art werden durch dieses Paket generiert, welches verschiedene Optionen gibt:

```
\LILLYxPATHxCONTROLLERS = source/Controllers
```

Bemerkung 15 – Standalone Boxen

Mit **VER 2.0.0** wurde die Listings-Integration als eigenes Paket **LIB LILLYxBOXES** etabliert, welches sich eigenständig über

```
\usepackage{LILLYxBOXES}
```

auch ohne das Verwenden der restlichen LILLY-Welt benutzen lässt.

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>

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

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

Um eine Fassng für jede Box zu generieren entspräche das:

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

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

```
\userput{_LILLY_BOXES_\LILLYxBOXxMODE}% File
{\lillyPathData}% User Path
{\LILLYxPATHxDATA/POIs}% Lilly Path
```

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 LILLYxBOXES` geladen. Sie befinden sich ebenfalls im eigenständigen Paket `LIB LILLYxCONTROLLERxBOX`.

- ◇ `\LoadLillyBoxMode{ModeName}` v2.0.0
Lädt den Box-Modus mit dem Bezeichner `ModeName`. Dieser Befehl erlaubt es auch in einer Definition eines eigenen Designs ein anderes zu Laden, darunter ebenfalls ein bereits definiertes oder eigenes, es werden dieselben Pfade überprüft.
- ◇ `\LILLYxBOXxMODE` v1.0.5
Hierrüber wird ausgewählt welcher der jeweiligen Boxmodi verwendet werden soll. Standardmäßig wird dieser Befehl von *Jake* gesetzt, aber natürlich kann dieser Befehl auch überschreiben werden.
- ◇ `\LILLYxBOXx<Bezeichner>xLock` v1.0.8
Enthält für die jeweilige Box, woran sich der Zähler orientieren soll. Enthält der Befehl `TRUE`, so wird ein Ungebundener Zähler verwendet. Wenn nicht definiert initialisiert durch:
- ◇ `\LILLYxBOXxHIGHLEVELxLOCK` v1.0.8
Enthält je nach Dokumenttyp entweder die höchste Hierarchie, an die ein Zähler gebunden werden kann oder `TRUE`. So erzeugt `TRUE` zum Beispiel den Zähler 4 und `section` in einem Dokument mit `\chapter: 1.13.4`.
- ◇ `\LILLYxBOXx<Bezeichner>xEnable` v1.0.8
Definiert, ob eine Box überhaupt angezeigt werden soll. Durch das Setzen auf `FALSE`, kann so eine Box aus dem Dokument genommen werden.
- ◇ `\LILLYxBOXx<Bezeichner>xBox` v1.0.8
Dieser Befehl besitzt (Stand `VER 2.0.0`) nicht für alle Boxbezeichner einen Effekt, steuert aber für bereits implementierte Boxen, ob diese durch das jeweilige Layout gesetzt werden sollen, oder ob die Box ohne die Box angezeigt werden soll. Die genaue Optik bestimmt wieder der jeweilige Modi.

Bemerkung 16 – Box-Kontrolle

Alle von Lilly generierten Boxen befinden sich mit `VER 2.0.0` in der Liste: `RegisteredBoxes` mit der Signatur `Name/Bezeichner`. So gehen die Konfigurationen wie folgt von statten:

```
1 \def\LILLYxBOXxBeweisxBox{FALSE} % Deaktiviert Beweisboxen
```

Hier die definierten Umgebungen in ihrer freien Wildbahn und Gestalt:

Definition 6.1 – Titel

moin

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

Definition 6.2 – Titel

moin

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

Bemerkung 17 – Titel

moin

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

Beispiel 6.1 – Titel

moin

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

Satz 6.4 – Titel

moin

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

Beweis 6.1 – Titel

moin

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

Lemma 6.1 – Titel

moin

```

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

```

Zusammenfassung 6.1 – Titel

moin

```

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

```

Aufgabe 4 – Titel

moin

```

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

```

Nicht richtig darstellbar aber weiter existiert:

```

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

```

◇ `env@task[opt-Addons]{Titel}{Punkte}` WAR Veraltet v1.0.0

Ein aus dem `eagleStudiPackage` stammendes Relikt, welches nur aus Kompatibilitätsgründen gehalten wird. Ebenso:

◇ `\DEF{Title}{Content}, \BEM{Titel}{Content}, ...` WAR Veraltet v1.0.0

Kompatibilitätsbefehle, der zur `eagleStudiPackage`-Zeit die Boxen gesetzt hat, nun allerdings die Daten an die jeweilige Umgebung weitergeben.

◇ `\inputUB{Name}{Nummer}{Pfad}, \inputUBS{Name}{Bezeichner}{Pfad}` v1.0.3

Binden eine Datei als Übungsblatt ein und erlaubt so, Übungsblätter in Mitschriften zu integrieren. Letzterer Befehl verwendet `uebungsblatt*`, verändert also nicht die Nummer, was bedeutet, dass auch Buchstaben oder anderes als Bezeichner möglich ist. *Diese werden, entsprechend der Regel von `uebungsblatt` nur angezeigt, sofern `\LLYxMODExEXTRA` den Wert `TRUE` enthält.*

Bemerkung 18 – Zugriff auf die Boxzähler

Der Zugriff auf die von Lilly unterhaltenen Boxzähler ist sehr Umständlich (Beispiel: `\thetcb@cnt@LILLYxBOXxDefinition`). Deswegen existiert das Hilfspaket `Lib LILLYxBOXxCOUNTER`, welches hierfür Kurzbefehle definiert: `\CTRxDf`, `\CTRxBEI`, `\CTRxBEM`, `\CTRxSAT`, `\CTRxBEW`, `\CTRxLEM` und `\CTRxZSM`. So liefert `\arabic\CTRxDf` : 2.

◇ `\RegisterBox[Box-Keys][Tikz-Keys]{Box-Name}{Title}{BoxID}`

v2.0.0

Registriert eine (neue oder alte) Box, die entsprechend Gesetzt werden kann. Die gezielte Verwendung dieses Befehls bedarf einiger Vorkenntnisse über das jeweilige Szenario. Es werden eine ganze Menge an Box-Keys gestattet, die auf einem anderen Verfahren persistiert werden (weswegen Box-gebundene Befehle mithilfe von `\noexpand` abgesichert werden müssen!). Alle so zur Verfügung stehenden Boxen werden durch `\RegisterBox` definiert.

Bezeichner	Typ	Standard	Beschreibung
name	<i>String</i>	noname	Name der Box
title	<i>String</i>	<name>	Titel der Box
boxcol	<i>Farbe</i>	black	Farbe für Links
preCode	<i>Code</i>		Befehle, die vor der Box gesetzt werden
inCode	<i>Code</i>		Befehle, die zu Beginn der Box gesetzt werden
outCode	<i>Code</i>		Befehle, die zu Ende der Box gesetzt werden
postCode	<i>Code</i>		Befehle, die nach der Box gesetzt werden
usestyle	<i>Box</i>	<Def>	Basisbox
emblem	<i>Code</i>		Titelverzierer
createlist	<i>Bool</i>	false	Erstelle Liste
customlist	<i>Bool</i>	false	Trage Box in Liste ein
boxenabled	<i>Bool</i>	true	Soll die Box angezeigt werden?
usebox	<i>Bool</i>	true	Box (true) oder das Plaindesign (false)?
lock	<i>Lock</i>	TRUE	Setzt die Zählersperre
listname	<i>String</i>	<name>	Name der Liste
listtext	<i>String</i>	<name>	Titel der Liste
listmen	<i>String</i>	NO	Mnemonic der Liste

Mit dem Registrieren einer Box, werden die folgenden Befehle für die jeweilige BoxID registriert. Sie werden expandiert, weswegen eine Absicherung mithilfe von `\noexpand` für

übergebene Befehle erfolgen sollte. Im Folgenden wird `<BoxID>` für die Befehle als Platzhalter verwendet:

◇ <code>\lillyxBOXx<BoxID>xName</code>	v2.0.0
◇ <code>\lillyxBOXx<BoxID>xTitle</code>	v2.0.0
◇ <code>\lillyxBOXx<BoxID>xBoxCol</code>	v2.0.0
◇ <code>\lillyxBOXx<BoxID>xPreCode</code>	v2.0.0
◇ <code>\lillyxBOXx<BoxID>xInCode</code>	v2.0.0
◇ <code>\lillyxBOXx<BoxID>xOutCode</code>	v2.0.0
◇ <code>\lillyxBOXx<BoxID>xPostCode</code>	v2.0.0
◇ <code>\lillyxBOXx<BoxID>xUseStyle</code>	v2.0.0
◇ <code>\lillyxBOXx<BoxID>xBoxEnabled</code>	v2.0.0
◇ <code>\lillyxBOXx<BoxID>xUseBox</code>	v2.0.0
◇ <code>\lillyxBOXx<BoxID>xEmblem</code>	v2.0.0
◇ <code>\lillyxBOXx<BoxID>xLock</code>	v2.0.0
◇ <code>\lillyxBOXx<BoxID>xListName</code>	v2.0.0
◇ <code>\lillyxBOXx<BoxID>xListText</code>	v2.0.0
◇ <code>\lillyxBOXx<BoxID>xListMen</code>	v2.0.0

Sie speichern den Wert der sich jeweils vermuten lässt. Weiter werden noch die beiden Booleschen Werte gespeichert, die entsprechend es Zustands TRUE und FALSE:


◇ <code>\lillyxBOXx<BoxID>xCreateList</code>	v2.0.0
◇ <code>\lillyxBOXx<BoxID>xCustomList</code>	v2.0.0

Die Modifikation dieser ermöglicht eine weitere manuelle Anpassung der Box, allerdings wird von einem direkten, modifizierendem Zugriff abgeraten. `\RegisterBox` legt weiter auch noch die entsprechende Umgebung auf Basis des Namens an. Gilt es eine bestehende Box zu modifizieren, so gilt es folgenden Befehl zu nutzen:

◇ <code>\TransformBox[*][new-args]{OldBoxID}{Box-title}[new-name] {NewBoxID}</code>	v2.0.0
---	--------

Dieser Befehl nimmt die Werte einer bereits bestehenden Box und Transformiert sie in eine neue Box. Sollte keine Box mit der entsprechenden ID existieren, so wird die BoxID als Name angenommen auf dessen Namen die Umbenennung geschieht. So lassen sich auch bereits bestehende Umgebungen, die bisher keine Box dargestellt haben, entsprechend modifizieren und als Box rekreieren. Wird der Stern gesetzt, so wird die alte Box nicht gelöscht, sondern lediglich eine neue Box kreiert.

6.1.3 Die Boxmodi

Generell muss einer der unten aufgeführten Modi keine einzige Box definieren überladen oder modifizieren, das geladene Default-Paket wird jeweils für nicht überladene Boxen die Standardboxen zur Verfügung stellen. Folgende Boxbezeichner sind hierbei von Relevanz. Das Default-Design stellt hierbei mit  die in der

Liste RegisteredBoxes aufgeführten Boxen zur Verfügung. An einer Normierung der aufgabe-Box wird gearbeitet: LILLYxBOXxDefinition, LILLYxBOXxBeispiel, LILLYxBOXxBemerkung, LILLYxBOXxSatz, LILLYxBOXxBeweis, LILLYxBOXxLemma, LILLYxBOXxZusammenfassung, LILLYxBOXxAufgabe, LILLYxBOXxAufgabexPLAIN und LILLYxBOXxUebungsblatt. In der Regel wird weiter noch ein Design generiert, so setzt zum Beispiel das Default-Design: LillyxBOXxDesignxDefault.

Default-Design

Mit VER 1.0.0 stellt dieses Design den Urvater dar. Bis VER 1.0.6 überarbeitet hier die finale Form:

Zusammenfassung 6.2

Wichtige Box - Yeah

Wir haben schon viel gelernt, zum Beispiel, dass der Apfel nicht weit vom Stamm fällt. Das ist aber eigentlich dann auch schon so das Einzige, was es wirklich zu lernen gilt im Kontext dieser wundervollen Boxwelt!

Erzeugt durch den bekannten Aufruf:

```
1 \begin{zusammenfassung}[Wichtige Box – Yeah]
2   %% ...
3 \end{zusammenfassung}
```

Auf Basis des Pakets tcolorbox definiert LILLY das Design LillyxBOXxDesignxDefault mit folgender Implementation:

```
1 \tcbset{LillyxBOXxDesignxDefault/.style={enhanced jigsaw,
2   pad before break*=2mm, pad after break=2mm, %
3   lines before break=4, before skip=0pt, boxrule = 0mm, %
4   toprule=0.5mm, bottomtitle=0.5mm,bottomrule=1.2mm, %
5   after skip=0pt, enlarge top by=0.2\baselineskip, %
6   enlarge bottom by=0.2\baselineskip, %
7   sharp corners=south, enforce breakable}%
8 }
```

Auf Basis dessen werden nun die einzelnen Boxumgebungen generiert. Hier exemplarisch die obige Zusammenfassung:

```
1 \DeclareTColorBox[auto counter]%
2   {LILLYxBOXxZusammenfassung}%
3   { 0{ } %% Title
4     0{Zusammenfassung \thetcbcounter~} %% TitlePrefix
5     0{ } %% tcb addonargs
6   }{%
7   LillyxBOXxDesignxDefault, %
8   colback=\LILLYxColorxZusammenfassung!5!white, %
9   colframe=\LILLYxColorxZusammenfassung, #3,%
10  title={\LILLYxDEFAULTxTYPESETxTITLE{#1}{#2}%
11         \ifx\LILLYxBOXxZusammenfassungxLock\true\\%
12         \else\\[-0.4\baselineskip]\fi}% spacing
13 }
```

Hierbei verwendet das ganze Paket den vermerkten `\LILLYxDEFAULTxTYPESETxTITLE`, der selbst wie folgt konstruiert ist:

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   { O{} O{Definition \thetcbcounter~} O{drop fuzzy shadow} }%
4   {LillyxBOXxDesignxDefault, colback=\LILLYxColorxDefinition!5!
5     white,%
6     colframe=\LILLYxColorxDefinition, #3,%
7     title={%
8       \begin{minipage}[t][\baselineskip][1]{\textwidth}%
9         \textbf{\textsc{{#2}}}\ \hfill {\textbf{#1}}%
10      \end{minipage}%
11    }%
12  }

```

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

```

1 \DeclareTColorBox{\LILLYxBOXxAufgabe}{O{} O{} O{}}{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 }

```


Beispiel 6.2 – Wie man ein eigenes Box-Design erzeugt

Im Folgenden finden wir das Default-Design einer Definition einfach hässlich und möchten es durch eine wunderschöne blaue Box ersetzen. Wir nennen das Design „quackbox“ und speichern es als `_LILLY_BOXES_quackbox.tex` im gleichen Ordner wie unser wundervolles Tex-Dokument. Weiter stört uns allgemein das Default-Design und wir wollen für unser Design gerne ALTERNATE als Grundlage nutzen. In die Definition unseres Boxdesigns schreiben wir also:

```
1 \LoadLillyBoxMode{ALTERNATE}% Laden des ALTERNATE-Designs
2 \RenewTCOLORBox[use counter from=LILLYxBOXxDefinition]{%
  LILLYxBOXxDefinition}%
3 { O{} O{} O{} }%
4 {#3, colframe=bondiBlue, title={#2#1},%
5 }
```

Das Tex-Dokument das wir nutzen möchten soll die article-Klasse und nicht Lilly nutzen, wir verwenden also nur die Bibliothek `LILLYxBOXES`. Die Nutzer-Definitionen sucht Lilly über den Pfad `\lillyPathData`. Wir schreiben also:

```
1 \documentclass{article}
2 \def\lillyPathData{.}% Suche im Dokumentordner
3 \def\LILLYxBOXxMODE{quackbox}%Nutze das gute Design
4 \usepackage{LILLYxBOXES}
5
6 \begin{document}
7   \begin{definition}[Hallo Welt]
8     Ich bin eine Definition
9   \end{definition}
10
11   \begin{bemerkung}[Hallo Welt]
12     Ich sehe wie eine Alternate-Bemerkung aus
13   \end{bemerkung}
14 \end{document}
```

Anstelle der Definition von `\LILLYxBOXxMODE` können wir, dank der Vordefinitionen in `LILLYxVANILLA` folgendes schreiben:

```
1 \documentclass{article}
2 \usepackage{LILLYxBOXES}
3 \begin{document}
4   \LoadLillyBoxMode{quackbox}
5   \begin{definition}[Hallo Welt]
6     Ich bin eine Definition
7   \end{definition}
8
9   \begin{bemerkung}[Hallo Welt]
10     Ich sehe wie eine Alternate-Bemerkung aus
11   \end{bemerkung}
12 \end{document}
```

In beiden Fällen erzeugen wir die folgenden Boxen:

Definition 0.1 Hallo Welt

Ich bin eine Definition

Bemerkung 0.1 – Hallo Welt

Ich sehe wie eine Alternate-Bemerkung aus

6.2 Info-Boxes

Diese Definitionen befinden sich in der Datei: `Controllers/LILLYxCONTROLLERxBOX`. Sie werden mit `VER 2.0.0` automatisch mit dem Einbinden von `LIB LILLYxBOXES` geladen. Sie befinden sich ebenfalls im eigenständigen Paket `LIB LILLYxBOXxINFOBOXES`.

Dieses Paket liefert gemeinsam mit `LIB LILLYxEMBLEMS` einige nützliche Boxen wie:



Dies ist eine Info-Box

Diese Info-Box kann ganz einfach mithilfe von folgendem Code erstellt werden:

```
1 \begin{infoBox}
2   Diese Info-Box kann ganz einfach mithilfe von folgendem Code ,
   erstellt werden:
3   % ....
4 \end{infoBox}
```

Jeweils existiert auch noch eine mit einem Stern markierte Umgebung die sich entsprechend in die jeweilige Margin des Dokuments einnistet. Da die Dokumentation ohne große Margin für etwaige Paragraphen konzipiert wurde, lässt die Optik hier selbstredend zu Wünschen übrig.^πDiese Box (inklusive klickbarem Marker im Text) wurde generiert durch:

```
1 \begin{mathBox*}{Hi}
2   $E=mc^2$
3 \end{mathBox*}
```

π Hi

$E = mc^2$

6.2.1 Wie es funktioniert

Gemeinsam mit `LIB LILLYxMARGIN` und `LIB LILLYxBOXxMARGIN` werden die Boxen für den Rand generiert, wobei diese sich für zweiseitige Dokumente selbstredend auch anpassen. Für die Boxen im Text wird `tcolorbox` verwendet. Eine neue Info-Box (inklusive „gesternter“-Umgebung) definiert sich einfach durch folgenden Befehl:

◇ `\NewInfoBox[InfoBox-Keys][tcb-Keys]{Name}`

v2.0.0

Für die InfoBox Keys gibt es wieder eine ganze Liste an Feldern, die konfigurierbar sind. Sie werden persistiert, wobei das Präfix „lillyxINFOBOXESx“ verwendet wird:

Bezeichner	Typ	Standard	Beschreibung
style	<i>enum</i> ^(a)	style/limerence	Design der Boxen
bgcolor	<i>Farbe</i>	MudWhite!75	Hintergrundsfarbe
bordercolor	<i>Farbe</i>	DebianRed	Rahmenfarbe
fgcolor	<i>Farbe</i>	Charcoal	Textfarbe
titlefont	<i>Code</i>	<code>\normalfont\bfseries</code>	Schrift des Titels
textfont	<i>Code</i>	<code>\normalfont</code>	Schrift des Textes
preCode	<i>Code</i>		Befehle, die vor der Box gesetzt werden
inCode	<i>Code</i>		Befehle, die zu Beginn der Box gesetzt werden
titleCode	<i>Code</i>	<code><long></code> ^(b)	Befehle, die nach dem Titel gesetzt werden
outCode	<i>Code</i>		Befehle, die zu Ende der Box gesetzt werden
postCode	<i>Code</i>		Befehle, die nach der Box gesetzt werden
emblem	<i>Code</i>		Emblem, welches gesetzt werden soll
marker	<i>Code</i>	<code>\textbf{!}</code>	Marker für Randnotiz

So wird zum Beispiel die codeBox wie folgt definiert:

```
1 \NewInfoBox[fgcolor={black},bgcolor={MudWhite!50},bordercolor={
  DarkOrchid},emblem={\btextEmblem\codeEmblem~},marker={\textbf{
  \faCode}}]{codeBox}
```

Es werden die folgenden Boxen vordefiniert: infoBox*ⁱ, warningBox*[!], errorBox*[⚡], mathBox*^π und codeBox*^{</>}. Natürlich jeweils auch mit den normalen Umgebungen:



Ich bin eine Information

Nunc sed pede. Praesent vitae lectus. Praesent neque justo, vehicula eget, interdum id, facilisis et, nibh. Phasellus at purus et libero lacinia dictum. Fusce aliquet. Nulla eu ante placerat leo semper dictum. Mauris metus. Curabitur lobortis. Curabitur sollicitudin hendrerit nunc. Donec ultrices lacus id ipsum.



Ich bin eine Warnung

Nunc sed pede. Praesent vitae lectus. Praesent neque justo, vehicula eget, interdum id, facilisis et, nibh. Phasellus at purus et libero lacinia dictum. Fusce aliquet. Nulla eu ante placerat leo semper dictum. Mauris metus. Curabitur lobortis. Curabitur sollicitudin hendrerit nunc. Donec ultrices lacus id ipsum.

^(a) Allowed: none, limerence, framed

^(b) `\leavevmode\smallskip\newline`



Hi

Hallo Welt



Hi

Hallo Belt



Hi

Hallo Geld



Hi

Hallo Wält



Hi

Hello
World

**Ich bin ein Error**

Nunc sed pede. Praesent vitae lectus. Praesent neque justo, vehicula eget, interdum id, facilisis et, nibh. Phasellus at purus et libero lacinia dictum. Fusce aliquet. Nulla eu ante placerat leo semper dictum. Mauris metus. Curabitur lobortis. Curabitur sollicitudin hendrerit nunc. Donec ultrices lacus id ipsum.

**Ich bin eine Mathe-Box**

Nunc sed pede. Praesent vitae lectus. Praesent neque justo, vehicula eget, interdum id, facilisis et, nibh. Phasellus at purus et libero lacinia dictum. Fusce aliquet. Nulla eu ante placerat leo semper dictum. Mauris metus. Curabitur lobortis. Curabitur sollicitudin hendrerit nunc. Donec ultrices lacus id ipsum.

**Ich bin eine Code-Box**

Pellentesque interdum sapien sed nulla. Proin tincidunt. Aliquam volutpat est vel massa. Sed dolor lacus, imperdiet non, ornare non, commodo eu, neque. Integer pretium semper justo. Proin risus. Nullam id quam. Nam neque. Duis vitae wisi ullamcorper diam congue ultricies. Quisque ligula. Mauris vehicula.

Weiter seien auch einmal die Stile veranschaulicht:

**Warnung**

Hallo Welt, na wie geht es dir? Ist super oder? Jaaaaa! Tihhi.

style/none

**Warnung**

Hallo Welt, na wie geht es dir? Ist super oder? Jaaaaa! Tihhi.

style/framed

**Warnung**

Hallo Welt, na wie geht es dir? Ist super oder? Jaaaaa! Tihhi.

style/limerence

Sowie: , und .

◇ `\dateBox{Datum/Text}`

v2.0.0

Dieser Befehl ist einfach nur exemplarisch entstanden um zu demonstrieren, wie die Boxen auch misshandelt werden können ☹. natürlich sollte eigentlich `\lillyxMarginxElement` verwendet werden: `\dateBox{\heute}`. Wobei die Definition wie folgt von statten geht:

```
1 \NewInfoBox[fgcolor={DarkMidnightBlue},bgcolor={MudWhite!0},,
  bordercolor={DarkOrchid},emblem={\faCalendar~},marker={},style/none,titleCode={}]{@dateBox}
2 \def\dateBox#1{\begin{@dateBox*}{#1}\end{@dateBox*}}
```

Wie bereits erwähnt, werden die Einstellungen für eine InfoBox persistiert. Hierzu werden folgende Befehle verwendet:

◇ `\lillyxINFOBOXESx<InfoBox>xDraw`

v2.0.0

◇ `\lillyxINFOBOXESx<InfoBox>xBgColor`

v2.0.0



Hi

Stups.



Hi

Stups.



Hi

Stups.

6. September 2019

◇ \lillyxINFOBOXESx<InfoBox>xFgColor	v2.0.0
◇ \lillyxINFOBOXESx<InfoBox>xBorderColor	v2.0.0
◇ \lillyxINFOBOXESx<InfoBox>xTitleFont	v2.0.0
◇ \lillyxINFOBOXESx<InfoBox>xTextFont	v2.0.0
◇ \lillyxINFOBOXESx<InfoBox>xPreCode	v2.0.0
◇ \lillyxINFOBOXESx<InfoBox>xInCode	v2.0.0
◇ \lillyxINFOBOXESx<InfoBox>xTitleCode	v2.0.0
◇ \lillyxINFOBOXESx<InfoBox>xOutCode	v2.0.0
◇ \lillyxINFOBOXESx<InfoBox>xPostCode	v2.0.0
◇ \lillyxINFOBOXESx<InfoBox>xEmblem	v2.0.0
◇ \lillyxINFOBOXESx<InfoBox>xMarker	v2.0.0

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 wird als `.jar`-Datei geliefert und lässt sich, eine vorhandene Installation von Java vorausgesetzt, durch das bloße ausführen installieren. Auf Linux kann dies zum Beispiel wie folgt von statten gehen:

```
java -jar jake.jar
```

Nach abgeschlossener Installation sollte das Terminal neu gestartet, oder die Konfigurationsdatei neu geladen werden, um Jake zur Verfügung zu stellen. Das bloße Ausführen von `jake` sollte nun eine Hilfe anzeigen, die über die jeweiligen Optionen aufklärt.

Bemerkung 19 – C++ Jake

Bis zur Version VER 1.0.9 war Jake in C++ geschrieben und benötigt deswegen eine andere 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 Lilly mit Jake installieren

Mit VER 2.0.0 liefert *Jake* stets eine Version von Lilly mit, die sich nach einem Update automatisch mit dem nächsten Start von *Jake* aktualisiert, sofern sie einmal installiert wurde:

jake install. Wird hier eine Frage nach verschiedenen Installationsoptionen gestellt, so siehe bei den **Entwicklerinformationen** oder wähle einfach die Installation der enthaltenen Variante (vermutlich Option 2). Die automatische Aktualisierung wird durch eine Ausgabe getreu [Die Lilly-Installation wurde aktualisiert.] ausgegeben. Mithilfe von **jake GUI** kann Jake auch über die Kommandozeile im Grafischen-Modus gestartet werden, allerdings empfiehlt sich hierfür das Verwenden des Eintrags im Anwendungsmenü.

7.1.4 Jake im Überblick

Hier werden zuerst die Vorzüge der Kommandozeile präsentiert, da die grafische Variante von Jake noch nicht sinnvoll ausgebaut ist und sich bisher lediglich zum editieren von Konfigurationsdateien eignet.

Kommandozeile

Im Regelfall, zur Kompilierung eines Dokuments, genügt es **Jake** mit dem jeweiligen Dokument aufzurufen:

```
jake <Dokumentname.tex>
```

Der Kompilierung können nun eine endlose Reihe an Einstellungen übergeben werden die jeweils mit einem „-“ anzuführen sind. Eine boolesche Einstellung kann so bereits umgeschaltet werden. So liefert:

```
jake dump -debug -debug -debug
```

Für die Einstellung „debug“ *true*. Eine „normale“ Einstellung, welche ein Argument fordert wird durch einen Doppelpunkt beendet:

```
jake dump -lilly-author: "Sonnenprophet_Hamsterbacke"
```

Liefert den entsprechenden Author für *lilly-author*. Dies lässt sich auch bei booleschen Ausdrücken, hier mit dem Setzen der Werte *true* und *false* erzeugen. Final gibt es noch Listen, die auch so zugewiesen werden können, allerdings durch das anfügen von *+*: auch erweitert werden können. So liefert:

```
jake dump -lilly-boxes: "DEFAULT" -lilly-boxes+: "ALTERNATE"
```

Den Wert „DEFAULT ALTERNATE“ für die Einstellung *lilly-boxes*, die Trennung der Elemente (Leerfeld) wird von Jake automatisch erkannt ist aber in der Regel auf Leerfelder normiert.

Hier die große (und hoffentlich vollständige) Liste aller möglichen Einstellungen. Ist der Standardwert zu lang, so wird er durch *...* gekürzt, wenn er abhängig ist, wird dies in der Bemerkung erklärt. Es gilt zu beachten, dass sich durch Konfigurationsdateien alle Einstellungen modifizieren lassen und somit auch die Standardwerte verändern:

Bezeichner	Typ	Defaultwert	Beschreibung
Version	<i>String</i>	[...]	Aktuelle Version von Jake
file	<i>String</i>	dummy.tex	Datei, um die es gehen soll
answer	<i>String</i>		Antwort, die, sofern nicht leer, auf alle Fragen die Jake stellt zuerst gegeben wird. Ein setzen auf „y“ entspricht der -y-Option von apt.

operation	<i>String</i>	help	Was Jake tun soll
debug	<i>Boolean</i>	false	Gibt an, ob Debug ausgegeben werden soll oder nicht
debug-filter	<i>String</i>	.*	Veraltet
path	<i>String</i>	./	Pfad zu Lilly
what	<i>String</i>		Zusatzargument für manche Operationen
install-path	<i>String</i>	\$HOME/texmf	Ziel Pfad der Installation
gepardrule-path	<i>String</i>		Pfade für Gepardregeln (durch „:“ getrennt)
autoconf	<i>Boolean</i>	true	Soll automatisch eine .conf-Datei gewählt werden?
comment-pattern	<i>String</i>	![^!]*!	Kommentarmuster
lilly-path	<i>String</i>	\$(dirname...	Pfad zur Lilly.cls
lilly-out	<i>String</i>	./\$(BASE...	Ausgabeordner der Tex-Datei?
lilly-in	<i>String</i>	./	Input-Pfad für Dateien
lilly-nameprefix	<i>String</i>		Namenspräfix für Ausgabedatei
lilly-boxes	<i>List</i>	DEFAULT	Boxen für den Kompiliervorgang
lily-modes	<i>String</i>	default	Modi für den Kompiliervorgang
lilly-complete	<i>Boolean</i>	true	Vollständige Dokumentvariante
lilly-complete-name	<i>String</i>	COMPLETE-	Präfix der vollständigen Version
lilly-print-name	<i>String</i>	PRINT-	Präfix der Druckversion
lilly-cleans	<i>List</i>	log aux ...	Dateiendungen die von autoclean gelöscht werden
lilly-autoclean	<i>Boolean</i>	true	Sollen Dateien automatisch gelöscht werden?
lilly-compiletimes	<i>String</i>	2	Wie oft soll kompiliert werden
lilly-vorlesung	<i>String</i>	NONE	Um welche Vorlesung handelt es sich?
lilly-semester	<i>String</i>	0	Das wievielte Semester ist es
lilly-n	<i>String</i>	42	Um das wievielte Übungsblatt handelt es sich?
lilly-show-boxname	<i>Boolean</i>	true	Soll der Boxname angezeigt werden?
lilly-layout-loader	<i>String</i>		Pfad zu den Layouts
lilly-external	<i>Boolean</i>	false	Soll versucht werden, Grafiken auszulagern?

<code>lilly-external-out</code>	<i>String</i>	<code>extimg</code>	Ausgabeordner für ausgelagert Grafiken
<code>lilly-author</code>	<i>String</i>	<code>Florian...</code>	Author des Dokuments
<code>lilly-author-mail</code>	<i>String</i>	<code>florian.s...</code>	Email-Adresse des Authors
<code>lilly-signatur-farbe</code>	<i>String</i>	<code>Leaf</code>	Farbe für das Highlighting
<code>lilly-bibtex</code>	<i>String</i>		Bibtex-Datei (ohne Endung)
<code>lilly-doctype</code>	<i>String</i>	<code>Mitschrieb</code>	Typ des Dokuments
<code>lilly-configs-path</code>	<i>String</i>		Pfad zur Lilly-Konfigurationsdatei (<code>\lillyPathConfig</code>)
<code>lilly-data-path</code>	<i>String</i>		Pfad zu generellen Daten (<code>\lillyPathData</code>)
<hr/>			
<code>jobcount</code>	<i>String</i>	<code>2</code>	Wie viele verschiedene Threads sollen im <code>multithreaded-compile</code> gleichzeitig betrieben werden?
<code>error-count</code>	<i>String</i>	<code>5</code>	Wie viele Fehler sollen in der Vorschau maximal angezeigt werden?
<code>mk-name</code>	<i>String</i>	<code>Makefile</code>	Veraltet, Name des Makefiles
<code>mk-path</code>	<i>String</i>	<code>./</code>	Veraltet, Pfad des Makefiles
<code>mk-use</code>	<i>Boolean</i>	<code>false</code>	Veraltet, soll das Makefile verwendet werden?

Zusätzlich kann anstelle der vorangestellten Option wie `dump` beziehungsweise der `.tex`-Datei auch eine Konfigurationsdatei angegeben werden. Auf sie wird **hier** mehr eingegangen. Bei einer solchen Angabe handelt es sich um eine Kurzform der jeweiligen Optionen `config` und `file_compile`. Ausführlich würde man also zum Kompilieren dieser Dokumentation schreiben:

```
jake file_compile -file: Lilly-Dokumentation.doc.tex
```

beziehungsweise, zum Verwenden der beiliegenden Konfigurationsdatei:

```
jake config -file: doc.conf
```

Oder eben die Kurzform:

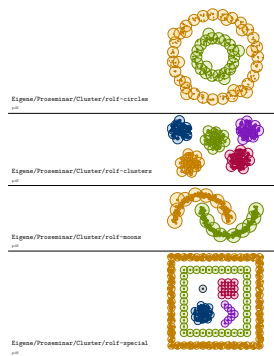
```
jake doc.conf
```

Die automatisch die Optionen entsprechend setzt.

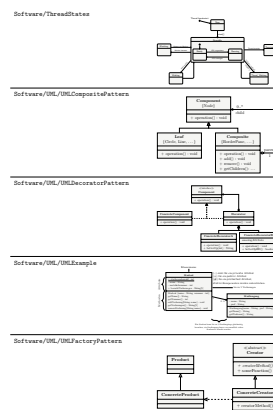
Als letzte wichtige Funktion sei noch `get` genannt, welche für das Paket `LIB LILLYxGRAPHICSxPROVIDER` aus `LIB LILLYxGRAPHICS` relevant ist. So liefert der Befehl:

```
jake get
```

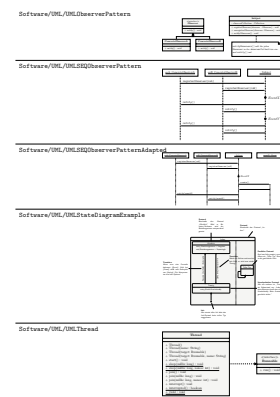
Eine PDF wie die folgende, die alle mit Lilly gelieferten Grafiken enthält:



13



33

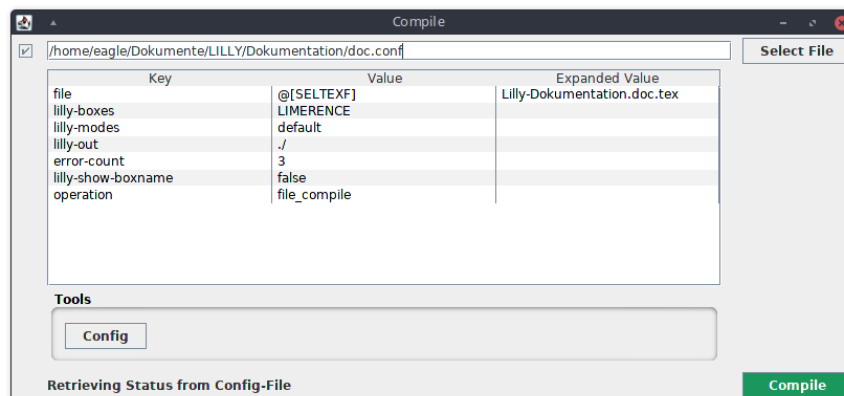


34

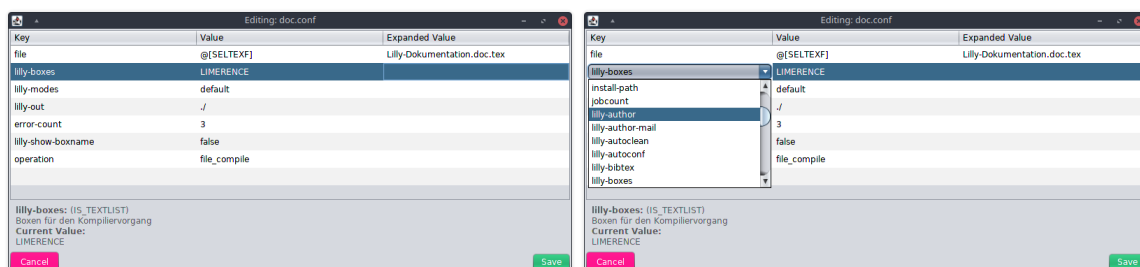
Angezeigt wurden hier übrigens die Seiten 13, 33 und 34 zum Kompilierzeitpunkt dieser Dokumentation.

Gui

Wie bereits angemerkt, ist die GUI von *Jake* noch nicht ansatzweise ausgereift. Der sich öffnende Hauptdialog erlaubt das Auswählen einer Latex- oder Konfigurationsdatei und zeigt die aus der Datei extrahierten Informationen inklusiver ihrer (sofern verschieden) extrahierten Werte an:



Nebst einem schönen Ausblick auf das was in der GUI-Welt noch so alles geschehen mag, bietet sich der Button Config an, der im Falle keine ausgewählten Konfigurationsdatei einfach auf einer neuen arbeitet:



Hier werden nicht nur einige Informationen zu den jeweiligen Einstellungen angezeigt, sondern auch Plausibilitätsprüfungen durchgeführt. Im Falle einer neuen Konfigurationsdatei kann beim Speichern ein entsprechendes Ziel ausgewählt werden.

7.1.5 Entwicklerinformationen

Wenn du an *Jake* oder Lilly mitentwickelst, oder einfach generell immer die aktuellste Version von Jake haben möchtest, so gilt es ein paar Schritte zu befolgen. Vorab: *Jake* wurde als Maven-Projekt angelegt, zum generieren der neusten Version ist also Maven für die jeweilige Plattform vonnöten (Beispiel: `sudo apt install maven`)

1. Kclone das Git-Repository (<https://github.com/EagleoutIce/LILLY>)
2. Navigiere in das Verzeichnis Jake/ im Repository
3. Generiere die neuste *Jake* -Version:

```
mvn clean install
```

4. Wähle aus (beachte die Angabe einer **Nutzerkonfiguration**):

stable Diese Installation wird unregelmäßig aktualisiert und entspricht der `jake.jar`, muss also nicht extra gebaut werden. Diese Variante empfiehlt sich für Tests und für die Entwicklung an Lilly, da so Fehler auf Seiten von *Jake* in der Regel ausgeschlossen werden können.

rolling Diese Installation wird nicht über das git-Repository synchronisiert sondern kann vom Nutzer bei Bedarf erzeugt werden. Hierzu einfach im Jake-Verzeichnis, den obigen Befehl (`mvn clean install`) ausführen. Im Unterverzeichnis `target/` befinden sich danach die `development-jake.jar`, die nach dem Ausführen mit jedem weiteren Kompilieren von *Jake* automatisch aktualisiert:

```
java -jar development-jake.jar
```

5. Bei der Installation von Lilly mit `jake install` ^(a) wird nun sicher eine Frage erscheinen, welche Variante von Lilly installiert werden soll. Während im **stable-tree** beide Optionen theoretisch verwendbar sind, so empfiehlt sich - auch um immer die aktuellste Lilly-Version zu haben, die Verlinkung der gefundenen Lilly-Instanz. Hier allerdings Vorsicht, da der Pfad zur `Lilly.cls` falsch sein kann. Der Pfad sollte in seiner Signatur auf `LILLY/Lilly/Lilly.cls` enden. Ist dies nicht der Fall, so muss die Option `lilly-path` angegeben werden, die den (am besten absoluten) Pfad zur `Lilly.cls` angibt.

7.2 Gepard

Die **Generator-Parser-Descriptor-Language** ist die Sprache, in der alle Konfigurationen und Erweiterungen von Jake formuliert sind. Im Kern des Parsen von Dokumenten steht der Tokenzier der erlaubte Zeichen und Zuweisungen unterscheidet, und vom Configurator erweitert wird. Dieses Konfigurationsmodul gestattet die Definition von **Konfigurationsdateien**, die durch einfache Zuweisungen die Einstellungen von *Jake* kontrollieren können.

^(a)Nicht vergessen, dass Terminal neu zu starten/die Konfigurationsdatei der Konsole neu einzulesen.

Darüber baut das namensgebende Gepard-Modul auf, welches verschiedene Boxen und damit verschiedene Erweiterungen gewährt, wobei bisher ein Verschachteln dieser Boxen nicht vorgesehen ist. Die einzelnen Module werden weiter **unten** beschrieben.

Bemerkung 20 – Kommentare

Ein Kommentar in Gepard wird in der Regel durch Ausrufezeichen markiert. Sollten diese allerdings verwendet werden müssen, so ist es möglich die Sequenz für Kommentare mithilfe von `comment-pattern` zu modifizieren:

```
1 debug      = true
2 ! debug = false !
3 what       = /* Noch kein Kommentar */
4 comment-pattern = /\*.*\*/
5 lilly-author = Hallo ! Sonne !
6 answer     = 42, /* Ich bin jetzt ein Kommentar :D */
```

Wir erhalten (mit `dump`) die Ausgabe für die modifizierten Werte:

```
comment-pattern : [/\*.*\*/]
debug           : [true]
lilly-author    : [Hallo ! Sonne !]
what           : [/* Noch kein Kommentar */]
answer         : [42,]
```

Keine mit Jake gelieferte Konfigurations- oder anderweitige Datei modifiziert die Syntax für einen Kommentar.

7.2.1 Konfigurationsdateien

Eine Konfigurationsdatei endet für gewöhnlich auf `.conf`, wobei diese Endung lediglich von der Autovervollständigung und der GUI anerkannt wird, allerdings keineswegs verpflichtend ist, *Jake* versucht jede Konfigurationsdatei entsprechend zu parsen. Erlaubt werden alle auch für *Jake* in der Kommandozeile verwendbaren **Einstellungen**, wobei zur Zuweisung hier `=` und `+=` anstelle von `:` und `+=` verwendet werden und kein „-“ angeführt wird. So kann eine Konfigurationsdatei wie folgt aussehen:

```
1 operation      = file_compile
2 file          = @[SELTEFX]
3 lilly-modes    = default
4 lilly-show-boxname = false
5 lilly-boxes    += LIMERENCE
6 lilly-out      = ./
7 error-count    = 3
```

Auch wenn hier zur Optik die Zuweisungen alle auf die gleiche Einrückung gesetzt wurden, so ist dies nicht zwingend und auch Tabs und Leerfelder haben im Verhältnis zur Zuweisung keine semantische Bedeutung und sind auch syntaktisch irrelevant. Das hier enthaltene `@[SELTEFX]` ist ein **Expandable**, welches über ein weiteres Gepard-Modul definiert wird. Dieses evaluiert zur ersten TeX-Datei die im Ordner gefunden wird, hat also den Vorteil,

dass diese Einstellung der konfigurationsdatei nicht immer wieder angepasst werden muss. Es gibt einige derartige Einstellungen.

Bemerkung 21 – Setzen von operation

Es ist zwangsläufig zu empfehlen die Einstellung `operation` auf das gewünschte Ziel zu überschreiben, da sonst die neue Datei (sofern überhaupt eine andere angegeben wurde) wieder mit der `config`-Operation ausgeführt wird, was im Zweifelsfall zu einer Endlosrekursion führen kann (diese wird von *Jake* natürlich erkannt und abgebrochen).

Weiter besitzt *Jake* die **Einstellung** `autoconf`, die eine Konfigurationsdatei bei der Wahl einer TeX-Datei auch automatisch auswählen kann sofern diese den gleichen Namen oder den Namen `jake.conf` trägt. So wird zum Beispiel beim Kompilieren von `Dokument.tex` automatisch die Datei `Dokument.conf` als Konfigurationsdatei geladen, sofern diese existiert. Analog würde die `jake.conf` gewählt werden, wenn sie existiert.

Bemerkung 22 – Standartkonfigurationsdateien

Jake selbst kommt mit der `jake_default.conf`, das ist eine Konfigurationsdatei, die für den aktuellen Build Einstellungen setzt, ohne jedesmal die in den `CoreSettings` vermerkten Einstellungen zu modifizieren. Diese Datei lässt sich theoretisch problemlos anpassen, davon wird allerdings stark abgeraten, da derartige Modifikationen mit einer Aktualisierung von *Jake* wieder überschrieben werden. Allerdings kann bei der Installation von *Jake* die Einstellung `path` gesetzt werden um eine Nutzerkonfiguration anzugeben. Diese wird von da an immer beim Starten von *Jake* eingelesen und verarbeitet:

```
java -jar jake.jar -path: /pfad/zu/meiner/Konfiguration.conf
```

Eine vorhandene Jake-Installation (auch zum Abändern dieses Pfades) kann mithilfe von `jake DEI` deinstalliert und mit `jake REI` reinstalliert werden, so kann auch bei einer bestehenden Installation von Jake mithilfe von:

```
jake REI -path: /neuer/pfad/zu/meiner/Konfiguration.conf
```

Der Pfad aktualisiert und mit:

```
jake REI
```

Der Nutzerpfad gelöscht werden. So Kann man mit dieser Konfiguration den Author aller Dokumente auf sich verändern:

```
1 ! Setze Autor für alle Dokumente: !
2 lilly-author           = Kalle Uweson
3 lilly-author-mail     = kalle.uweson@hotmail.waffle
4 ! Verstecke standartmäßig den Boxnamen: !
5 lilly-show-boxname    = false
6 ! Setze standardbox auf ALTERNATE !
7 lilly-boxes           = ALTERNATE
```

Aktuell wird überlegt, ob bei der Instalation direkt nach wichtigen Daten wie dem Namen gefragt wird.

7.2.2 Gepard Module im Allgemeinen

Gepardmodule werden in einer Datei als Box präsentiert. Eine Datei kann so etliche verschiedene Boxen und damit Konfigurationen für verschiedene Module halten und verarbeiten. Das grundlegende Gepard-Modul kann in einer (üblicherweise auf `.gpd` endenden) Datei wie folgt dargestellt werden:

```
1 BEGIN <modul>
2   <Modulspezifikation>
3 END
```

In die jeweiligen Start- und Endzeilen können beliebige Zeichen zur Übersicht Platziert werden, sie werden verworfen, was zum Beispiel folgende Spezifikation genauso valide macht:

```
1 BEGIN <modul>:
2   <Modulspezifikation>
3 END;
```

Die Spezifikation besteht in der Regel aus Konfigurationsähnlichen Zuweisungen, die je nach Modul eine unterschiedliche semantische Bedeutung haben. Die gewünschten Konfigurationen können über die Einstellung `gepardrule-path` gesetzt werden, wobei die Pfade durch einen „:“ getrennt sind. Bisher muss vom Nutzer die Existenz der zugrundeliegenden Dateien gewährleistet werden.

7.2.3 Buildrules

Buildrules definieren den Modus in dem das Dokument kompiliert wird. So definieren sie den `print` und den `default` Modus. Die Box trägt den Namen `buildrule` und muss einen Namen, einen Anzeigenamen und einen Modus definieren. Im Folgenden die Definition des `default`-Modus, die Kommentare sollten die Anforderungen zu Genüge erklären:

```
BEGIN buildrule: ! Der Doppelpunkt ist optional. Ich mag ihn, man braucht ihn nicht !

! Das Einrücken _und_ die Leerfelder sind optional. !
! Allerdings sollten erstmal nur Leerfelder verwendet werden !
! Mit X sind Zuweisungen markiert die verpflichtend sein sollen (aber nicht sind) !

!X! name                = default      ! buildrule name für lilly-modes !
!X! display-name        = Standard     ! Anzeigename (Standard-Version) !
!X! lilly-mode           = default      ! Welcher Modus soll an Lilly übergeben werden? !
!                               ! Info: Diese können noch nicht frei konfiguriert werden !

complete                = false       ! Keine complete-Version !
complete-prefix         = c_          ! Bezeichner wenn complete !
nameprefix              = MY-DEFAULT- ! Weicht vom normalen default ab !
lilly-complete-prefix   = COMPLETE-   ! Namenszusatz wenn complete Version (Default: COMPLETE-)!
lilly-loader            = \input{$(INPUTDIR)$(TEXFILE)}
! Diese Funktion ist advanced und beschreib die Einbinderoutine – einfach
! ignorieren !

END; ! Semikolon wieder nicht nötig, aber ich mag es :D !
```

Jeder so definierte Modus steht in den Einstellungen für `lilly-modes` zur Verfügung. Auch wenn sie bisher eher eingeschränkt agieren können, so bieten sie bereits einiges an Flexibilität.

7.2.4 Expandables

Die hier definierten Variablen können überall in Einstellungen oder anderen Gepardrule-Files verwendet werden. Abgesehen von einer rekursiven Definition ist alles gestattet. Jake

definiert bereits eine Reihe an Expandables, ein paar davon greifen auf Shell-Befehle zurück, was aus Sicherheitsgründen sonst nicht gestattet ist (im Klartext: Auch wenn es vordefinierte Expandables gibt die auf Shell-Befehle zurückgreifen, kann kein manuell definiertes Expandable eigene Shell-Befehle initiieren). Im Folgenden sind jeweils nur ihre Bezeichner angegeben, jedes Expandable kann durch `$[<Name>]` und `${<Name>}` angegeben werden um zum Zielwert zu evaluieren:

TEXTFILE	Expandiert zum vollen Bezeichner TeX-Datei
BASENAME	Expandiert zum Namen der TeX-Datei ohne Endung
FINALNAME	Expandiert zum Namen nach der Generierung (nur sofern im Kontext klar vorhanden)
LOGFILE	Expandiert zum Pfad der Logdatei
PDFFILE	Expandiert zum Namen der PDF-Datei
LATEXARGS	Expandiert zu den Latex-Argumenten (<code>-shell-escape, ...</code>)
OUTPUTDIR	Expandiert zum Ausgabeordner
INPUTDIR	Expandiert zum Quellordner
BOXMODES	Expandiert zu den Boxmodi
CLEANTARGETS	Expandiert zu den zu löschenden Endungen
SIGNATURECOL	Expandiert zur Signaturfarbe
AUTHOR	Expandiert zum Author
AUTHORMAIL	Expandiert zur Email-Adresse des Autors
NAMEPREFIX	Expandiert zum Namenspräfix
SEMESTER	Expandiert zur Semesterzahl
VORLESUNG	Expandiert zur Vorlesung
LILLY_CONFIGS_PATH	Expandiert zum Pfad der Konfigurationen
LILLY_DATA_PATH	Expandiert zum Pfad der Daten
N	Expandiert zur Übungsblattnummer
JOBCOUNT	Expandiert zur Maximalen Jobanzahl
_LILLYARGS	Expandiert zu den Argumenten für Lilly
_C	Expandiert zu einem wundervollen Komma ☺

HOME	Expandiert zum Homeverzeichnis
TRUE	Expandiert zu „ <i>true</i> “
FALSE	Expandiert zu „ <i>false</i> “
S_TRUE	Expandiert zur Jake-Definition von <i>true</i> („ <i>true</i> “)
S_FALSE	Expandiert zur Jake-Definition von <i>false</i> („ <i>false</i> “)

Es existieren noch einer Reihe besonderer Expandables, die entweder Shell Befehle beinhalten, oder außerhalb des direkten Dokumentenkontext steht. Sie besitzen die Signatur `@[<Name>]` und sind in der Regel lazy:

JAKEVER	Expandiert zur Jake Version
SELTEXF	Expandiert zu einer TeX-Datei des Verzeichnisses
SELCONF	Expandiert zu einer <code>.conf</code> -Datei des Verzeichnisses
GITHUB	Expandiert zum Github-Link des Repositories
CONFPATH	Expandiert zum Pfad der Nutzerkonfiguration
AUTONUM	Expandiert zu einer Zahl im Dateinamen, sofern dieser eine Zahl enthält, sonst 42
WAFFLE	Expandiert zur „GIVE ME THAT WAFFLE“ und wird für Tests verwendet
JAKECDATE	Veraltet, ist zum Kompiledatum der C++-Version expandiert
JAKECTIME	Veraltet, ist zum Kompilezeitpunkt der C++-Version expandiert

Die Definition eines Expandables ist relativ einfach, jede Zuweisung der Box wird als Expandable zur Verfügung gestellt:

```
1 BEGIN expandable:
2     SUPERWAFFEL = Ist Wichtig
3     S_TRUE      = FALSE ! Tihihhi !
4     S_FALSE     = TRUE ! höhöhöhö !
5     SuperHome   = ${HOME}/Tolle Welt/${TRUE}
6     LayoutConfig = @[CONFPATH]/Layout
7 END;
```

Sie lassen sich normal durch `${<name>}` und `$(<name>)` erweitern.

7.2.5 Hooks

Hooks sind etwas tolles ☺, sie können während des Kompilierprozesses Shell-Befehle ausführen und so Aktionen übernehmen wie das verschieben von Dateien oder dem Anstoßen weiterer Kompilierprozesse. Sie sind es auch, die beim Kompilieren den aktuellen Stand sowie die Lokalität des Logfiles ausgeben. Eine Hook besteht aus den folgenden Komponenten:

```
1 BEGIN hook :
2 !X!  name      =      in0-hook ! :D !
3 !X!  type      =      IN0
4      body      = echo "Hallo Welt – will it break?"
5      on-failure =      ! nothing at all !
6      on-success = ""    ! still nothign !
7 END;
```

Der Typ (type) einer Hook, kann die folgenden Bezeichner annehmen:

PRE	Wird vor dem Kompilieren ausgelöst
IN#	Führt, von 0 beginnend die Hook nach dem #-Kompiliervorgang aus
POST	Wird nach dem Kompilieren ausgelöst
ALL	Wird jedesmal ausgelöst

Wird die Hook ausgelöst, so wird der body in der entsprechenden Shell des Betriebssystems ausgeführt. Im Falle einer geglückten Operation wird on-failure im Fehlerfall wird on-success ausgeführt.

7.2.6 Name Maps

Um Faul bleiben zu können, wurde das nmap-Modu kreiert, welches für den Namen eines Dokuments gewisse Trigger generieren kann, die verschiedene Einstellungen setzen können. So existieren eine Reihe an Name Maps, die im Falle eines entsprechenden Dokumentnamens die jeweilige Einstellungen übernehmen:

PDP	pdp,PdP,PDP,[Pp]aradigmen[\\ \\\-]]?([Dd]er[\\ \\\-])?[Pp]rogrammierung	Setzt das Semester auf 2 und die Vorle- sung auf PDP
GDBS	gdbbs,GdBS,GDBS,[Gg]rundlagen[\\ \\\-]]?([Dd]er[\\ \\\-])?[Bb]etriebssysteme	Setzt das Semester auf 2 und die Vorle- sung auf GDBS
ANA1	ana1,ANA1,[Aa]nalysis[\\ \\\-]?1	Setzt das Semester auf 2 und die Vorle- sung auf ANA1
PVS	pvs,PvS,PVS,[Pp]rogrammierung[\\ \\\-]]?([Vv]on[\\ \\\-])?[Ss]ystemen	Setzt das Semester auf 2 und die Vorle- sung auf PVS

GDRA	<code>[Gg][Dd][Rr][Aa],[Gg]rundlagen[\\ \\-] ?([Dd]er[\\ \\-])?[Rr]echnerarchitektur</code>	Setzt das Semester auf 1 und die Vorlesung auf GDRA
EIDI	<code>[Ee][Ii][Dd][Ii],[Ee]inführung[\\ \\-] ?([Ii]n[\\ \\-]?)([Dd]ie[\\ \\-])?[Ii]nformatik</code>	Setzt das Semester auf 1 und die Vorlesung auf EIDI
FG	<code>[Ff][Gg],[Ff]ormale[\\ \\-] ?[Gg]rundlagen</code>	Setzt das Semester auf 1 und die Vorlesung auf FG
LA	<code>LA,LAI,[Ll]ineare[\\ \\-]?[Aa]lgebra</code>	Setzt das Semester auf 1 und die Vorlesung auf LA I
ÜB	<code>UB,uebungsblatt,[Üü]bungsblatt,ÜB</code>	Setzt den Modus auf „uebungsblatt“

Ein `nmap` braucht einen Namen und ein Pattern, mit dem er auslöst. Alle weiteren Einstellungen die Übergeben werden sind die einer **Konfigurationsdatei** und damit der **Einstellungen** von Jake die so übernommen werden. Hier ein Beispiel:

```

1 BEGIN nmap:
2     name           = Paradigmen—der—Programmierung
3     patterns       = GDBS,pdp,PDP,PdP
4
5     lilly—author    = Schlingelwingel
6
7     lilly—vorlesung = PDP!!
8     lilly—semester  = 2!!
9 END;
```

8

AUSSICHT

DAS WUNDER DER SCHÖPF... EVOLUTION ☺

8.1 Todos

8.1.1 Visuels

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.

8.1.2 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

8.1.3 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

8.1.4 Road to CTAN

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

8.1.5 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.

8.1.6 Weitere

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

ANHANG

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

9.1 Version VER 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 T_EX 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 23 – 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^AT_EX

VER 1.0.2

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

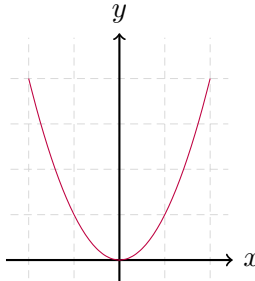
Die Anführungszeichen dienen hier und in anderen Codebeispielen lediglich zur Übersicht!

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) \> 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=0.6,domain=-2:2] 2 \plotline[purple]{\x}{\x*\x}; 3 \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:

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`

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.

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 24 – 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 25 – make

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

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

STICHWORTVERZEICHNIS

Ä's Ö's Ü's	
<code>env@<Sprache></code> <small>(v1.0.9)</small>	34
<code>env@<Sprache>*</code> <small>(v1.0.9)</small>	34
<code>\<name></code> <small>(v2.0.0)</small>	25

A	
<code>\abs</code> <small>(v1.0.9)</small>	9
<code>\ampelG</code> <small>(v1.0.2)</small>	17
<code>\arccot</code> <small>(v1.0.8)</small>	10

B	
<code>\B</code> <small>(v1.0.3)</small>	10
<code>\b<Sprache></code> <small>(v1.0.9)</small>	33
<code>\btextEmblem</code> <small>(v2.0.0)</small>	26

C	
<code>\c<Sprache></code> <small>(v1.0.9)</small>	33
<code>\ceil</code> <small>(v2.0.0)</small>	13
<code>\codeEmblem</code> <small>(v2.0.0)</small>	24
<code>\crossAT</code> <small>(v1.0.1)</small>	11

D	
<code>\das</code> <small>(v1.0.3)</small>	8
<code>\dateBox</code> <small>(v2.0.0)</small>	56
<code>\DEF</code> <small>(v1.0.0)</small>	48
<code>\det</code> <small>(v1.0.3)</small>	9
<code>\dif</code> <small>(v2.0.0)</small>	10

E	
<code>env@egraph</code> <small>(v2.0.0)</small>	15
<code>\enum</code> <small>(v1.0.0)</small>	11
<code>\epsilon</code> <small>(v1.0.3)</small>	10
<code>\errorEmblem</code> <small>(v2.0.0)</small>	24

F	
<code>\folge</code> <small>(v1.0.7)</small>	12

G	
<code>\gdw</code> <small>(v1.0.7)</small>	12

<code>\getGraphics</code> <small>(v2.0.0)</small>	22
<code>\getGraphicsPath</code> <small>(v2.0.0)</small>	23
<code>\getPrerendered</code> <small>(v2.0.0)</small>	23
<code>\gitRAW</code> <small>(v1.0.0)</small>	38
<code>env@graph</code> <small>(v1.0.8)</small>	13
<code>\graphdot</code> <small>(v1.0.2)</small>	18
<code>\graphPOI</code> <small>(v1.0.4)</small>	19

H	
<code>\Hcolor</code> <small>(v1.0.9)</small>	29

I	
<code>\i</code> <small>(v1.0.1)</small>	10
<code>\i<Sprache></code> <small>(v1.0.9)</small>	33
<code>\Im</code> <small>(v1.0.2)</small>	9
<code>\inf</code> <small>(v1.0.6)</small>	9
<code>\infoEmblem</code> <small>(v2.0.0)</small>	24
<code>\inputUB</code> <small>(v1.0.3)</small>	48
<code>\isLanguageLoaded</code> <small>(v2.0.0)</small>	35
<code>\isLanguageNameLoaded</code> <small>(v2.0.0)</small>	35
<code>\isRuntimeLoaded</code> <small>(v2.0.0)</small>	43

J	
<code>\join</code> <small>(v2.0.0)</small>	11

L	
<code>\LillyNewLstEnvironCore</code> <small>(v2.0.0)</small>	40
<code>\LillyNewLstEnvironPlain</code> <small>(v2.0.0)</small>	40
<code>\LillyNewLstEnvironPresent</code> <small>(v2.0.0)</small>	40
<code>\LILLYxBOXx<Bezeichner>xBox</code> <small>(v1.0.8)</small>	46
<code>\LILLYxBOXx<Bezeichner>xEnable</code> <small>(v1.0.8)</small>	46
<code>\LILLYxBOXx<Bezeichner>xLock</code> <small>(v1.0.8)</small>	46
<code>\lillyxBOXx<BoxID>xBoxCol</code> <small>(v2.0.0)</small>	50
<code>\lillyxBOXx<BoxID>xBoxEnabled</code> <small>(v2.0.0)</small>	50
<code>\lillyxBOXx<BoxID>xCreateList</code> <small>(v2.0.0)</small>	50
<code>\lillyxBOXx<BoxID>xCustomList</code> <small>(v2.0.0)</small>	50

T	
<code>env@task</code> ^(v1.0.0)	48
<code>\textEmblem</code> ^(v2.0.0)	26
<code>\TransformBox</code> ^(v2.0.0)	50
<code>\trenner</code> ^(v1.0.0)	12

V	
<code>\val</code> ^(v1.0.8)	10
<code>\VRule</code> ^(v1.0.4)	12

W	
<code>\warningEmblem</code> ^(v2.0.0)	24
<code>env@wgraph</code> ^(v1.0.8)	15

X	
<code>\x</code> ^(v1.0.2)	13
<code>\xa</code> ^(v1.0.1)	11
<code>\xmark</code> ^(v2.0.0)	13

Y	
<code>\ymark</code> ^(v2.0.0)	13