LilyPond

Das Notensatzprogramm

Handbuch zum Lernen

Das LilyPond-Entwicklerteam

Für LilyPond Version 2.11.58

Inhaltsverzeichnis

Vorwort

Es muss wohl während einer Probe des EJE (Eindhovens Jugendorchester) etwa 1995 gewesen sein, als Jan, einer der schrägen Bratschisten, Han-Wen, einem der verstimmten Hornisten, von seinem großartigen neuen Projekt erzählte, an dem er gerade arbeitete. Es sollte ein automatisiertes System für den Notensatz werden (um genauer zu sein, es war MPP, ein Präprozessor für MusiXTeX). Zufällig wollte Han-Wen gerade einige Stimmen einer Partitur ausdrucken, und so schaute er sich das Programm an und war schnell begeistert. Man entschied sich, dass MPP in eine Sackgasse führte, und nach vielem Philosophieren und hitzigem E-Mail-Austausch begann Han-Wen mit LilyPond 1996. Dieses Mal wurde Jan mitgerissen von Han-Wens neuem Projekt.

Die Entwicklung eines Computerprogramms erinnert in vielem an das Erlernen eines Musikinstrumentes. Am Anfang macht es Spaß herauszufinden, wie alles funktioniert und alles, was man noch nicht kann, stellt eine Herausforderung dar. Nach der ersten Begeisterung muss man jedoch viel üben. Tonleitern und Etüden können furchtbar langweilig sein, und wenn keine Ermunterung von anderen – Lehrern, Dirigenten oder dem Publikum – kommt, ist es oft eine große Versuchung, einfach aufzuhören. Aber man macht weiter, und langsam wird das Instrument zu einem Teil des eigenen Lebens. An manchen Tagen geht alles wie von selbst und es macht Spaß, an anderen Tagen ist es nur Arbeit, aber man macht trotzdem weiter, jeden Tag.

Die Arbeit an LilyPond kann genauso wie das Spiel eines Instruments sehr langweilig sein, und manchmal kommt es vor lauter Fehlern so vor, als stapfe man durch einen Morast. Trotzdem ist die Arbeit schon Teil unseres Lebens geworden und wir machen einfach weiter. Die wahrscheinlich wichtigste Motivation ist wohl, dass unser Programm wirklich nützlich ist. Wenn wir im Internet surfen, finden wir viele Leute, die LilyPond benutzen und damit außerordentlich beeindruckende Partituren erstellen. Das zu sehen fühlt sich auf angenehme Weise etwas unwirklich an.

Unsere Stimmung heben aber nicht nur die Benutzer unseres Programmes, sondern auch die vielen Menschen, die uns helfen, indem sie Vorschläge einbringen, auf verschiedene Art an LilyPond mitwirken oder Fehlerberichte schicken. Ihnen allen möchten wir hier Dank sagen!

Musik spielen und Musik zu Papier zu bringen ist mehr als eine nette Analogie. Zusammen zu programmieren macht viel Spaß und Menschen helfen zu können ist sehr zufriedenstellend, aber letzten Endes geht es uns darum, durch dieses Programm unsere Liebe zur Musik auszudrücken. Wir hoffen, Sie können viele schöne Partituren damit setzen!

Han-Wen und Jan

Utrecht/Eindhoven, Niederlande, Juli 2002.

1 Einleitung

Dieses Kapitel stellt dem Leser die Idee hinter LilyPond und die Dokumentation von LilyPond vor.

1.1 Hintergrund

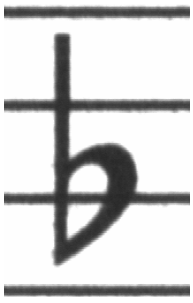
Dieser Abschnitt behandelt die allgemeinen Ziele und die Architektur von LilyPond.

Notensatz

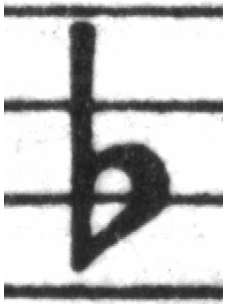
Die Kunst des Notensatzes wird auch als Notenstich bezeichnet. Dieser Begriff stammt aus dem traditionellen Notendruck. Noch bis vor etwa 20 Jahren wurden Noten erstellt, indem man sie in eine Zink- oder Zinnplatte schnitt oder mit Stempeln schlug. Diese Platte wurde dann mit Druckerschwärze versehen, so dass sie in den geschnittenen und gestempelten Vertiefungen blieb. Diese Vertiefungen schwärzten dann ein auf die Platte gelegtes Papier. Das Gravieren wurde vollständig von Hand erledigt. Es war darum sehr mühsam, Korrekturen anzubringen, weshalb man von vornherein richtig schneiden musste. Es handelte sich dabei um ein sehr spezialisiertes Handwerk.

Heutzutage wird fast alle gedruckte Musik von Computern erstellt. Das hat einige deutliche Vorteile: Drucke sind billiger als die gravierten Platten und der Computersatz kann per E-Mail verschickt werden. Leider hat der intensive Einsatz des Computers die graphische Qualität des Notensatzes vermindert. Mit dem Computer erstellte Noten sehen langweilig und mechanisch aus, was es erschwert, von ihnen zu spielen.

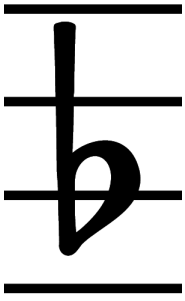
Die Abbildung unten illustriert den Unterschied zwischen traditionellem Notensatz und einem typischen Computersatz. Das dritte Bild zeigt, wie LilyPond die Formen des traditionellen Satzes nachahmt. Das linke Bild zeigt ein eingescanntes b-Vorzeichen aus einer 2000 herausgegebenen Edition. Das mittlere Bild zeigt das b-Vorzeichen der selben Musik aus einer handgestochenen Bärenreiter-Ausgabe. Das linke Bild zeigt die typischen Makel des Computer-Satzes: Die Notenlinien sind sehr dünn, die Schwärze des Vorzeichens entspricht den dünnen Linien und hat eine gerade Form mit scharfen Ecken und Kanten. Im Gegensatz dazu hat das Bärenreiter-Vorzeichen dicke, gerade zu sinnlich rundliche Formen. Unser Symbol für das Vorzeichen hat neben anderen auch dieses b als Vorbild. Es ist abgerundet und passt zu unseren Notenlinien, die sehr viel dicker sind als die der entsprechenden Computer-Ausgabe.



Henle (2000)

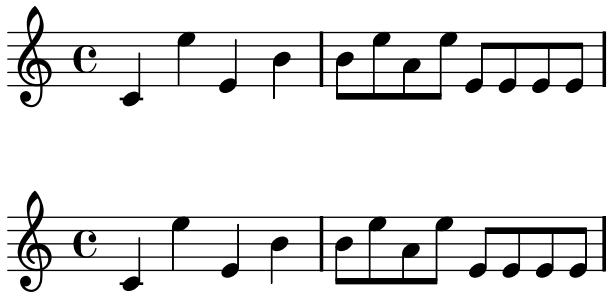


Bärenreiter (1950)



LilyPond
Feta-Schriftart
(2003)

Die Verteilung der Noten innerhalb des Taktes sollte ihrer Dauer entsprechen. Moderne Partituren zeigen diese Verhältnisse jedoch mit einer mathematischen Präzision, die nur sehr schlechte Ergebnisse bringt. Im nächsten Beispiel ist ein Motiv zweimal gesetzt: einmal mit den exakten mathematischen Längenverhältnissen, dann mit kleinen Korrekturen. Welches von beiden ist mit dieser Korrektur gesetzt?



In diesem Ausschnitt kommen nur Viertel vor, Noten, die in einem gleichmäßigen Rhythmus gespielt werden. Die Abstände sollten das widerspiegeln. Leider lässt uns aber das Auge im Stich: es beachtet nicht nur den Abstand von aufeinander folgenden Notenköpfen, sondern auch den ihrer Hälse. Also müssen Noten, deren Hälse in direkter Folge zuerst nach oben und dann nach unten ausgerichtet sind, weiter auseinander gezogen werden, während die unten/oben-Folge engere Abstände fordert, und das alles auch noch in Abhängigkeit von der vertikalen Position der Noten. Das obere Beispiel ist mit dieser Korrektur gesetzt, das untere ohne. In letzterem Fall bilden sich für das Auge bei unten/oben-Folgen Notenklumpen mit schmalen Abständen zwischen den Notenhälsen.

Musiker sind üblicherweise zu zu konzentriert, die Musik aufzuführen, als das Aussehen der Noten zu studieren; und diese Beschäftigung mit typographischen Details mag akademisch wirken. Das ist sie aber nicht. Unser Beispielstück hat einen monotonen Rhythmus, und wenn alle Zeilen gleich aussehen, wird das Notenblatt zu einem Labyrinth. Wenn der Spieler auch nur einmal wegschaut oder kurze Zeit unkonzentriert ist, findet er nicht mehr zurück zu der Stelle, an der er war.

Der dichtere Eindruck, den die dickeren Notenlinien und schwereren Notationssymbole schaffen, eignet sich auch besser für Noten, die weit vom Leser entfernt stehen, etwa auf einem Notenständer. Eine sorgfältige Verteilung der Zwischenräume erlaubt es, die Noten sehr dicht zu setzen, ohne dass die Symbole zusammenklumpen. Dadurch werden unnötige Seitenumbrüche vermieden, sodass man nicht so oft blättern muss.

Dies sind die Anforderungen der Typographie: Das Layout sollte schön sein – nicht aus Selbstzweck, sondern um dem Leser zu helfen. Für Aufführungsmaterial ist das umso wichtiger, denn Musiker haben eine begrenzte Aufmerksamkeit. Je weniger Mühe nötig ist, die Noten zu erfassen, desto mehr Zeit bleibt für die Gestaltung der eigentlichen Musik. Das heißt: Gute Typographie führt zu besseren Aufführungen!

Die Beispiele haben gezeigt, dass der Notensatz eine subtile und komplexe Kunst ist und gute Ergebnisse nur mit viel Erfahrung erlangt werden können, die Musiker normalerweise nicht haben. LilyPond stellt unser Bemühen dar, die graphische Qualität handgestochener Notenseiten ins Computer-Zeitalter zu transportieren und sie für normale Musiker erreichbar zu machen. Wir haben unsere Algorithmen, die Gestalt der Symbole und die Programm-Einstellungen darauf abgestimmt, einen Ausdruck zu erzielen, der der Qualität der alten Editionen entspricht, die wir so gerne betrachten und von denen wir gerne spielen.

Automatisierter Notensatz

Wie sollen wir also jetzt die Typographie anwenden? Wie können wir erwarten, dass wir in der Lage wären, ein Programm zu schreiben, dass den Beruf des Notenstechers ersetzt, wo dieser doch mehr als zehn Jahre braucht, um ein Meister zu werden?

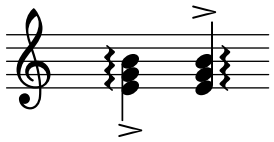
Wir können es tatsächlich nicht! Da Typographie allein durch das menschliche Auge bestimmt ist, kann der Mensch nicht ersetzt werden. Aber sehr viel mechanische Arbeit kann automatisiert werden. Indem etwa LilyPond die üblichen Situationen kennt und bewältigt, können die restlichen Fehler von Hand beseitigt werden. Das ist schon ein großer Fortschritt im Vergleich zu den existierenden Programmen. Und mit der Zeit können immer mehr Fälle automatisiert werden, so dass immer weniger Eingriffe von Hand notwendig werden.

Als wir anfingen, haben wir LilyPond vollständig in der Programmiersprache C++ geschrieben. Das hieß, dass der Funktionsumfang des Programms vollständig durch die Programmierer festgelegt war. Das stellte sich aus einer Reihe von Gründen als unzureichend heraus:

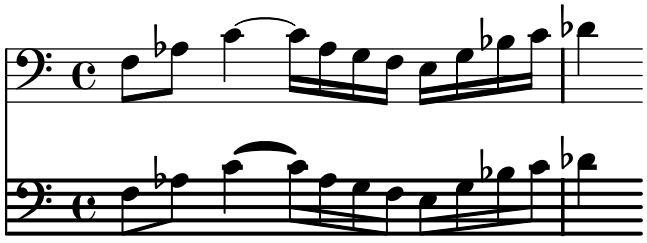
- Wenn LilyPond Fehler macht, muss der Benutzer die Einstellungen ändern können. Er muss also Zugang zur Formatierungsmaschinerie haben. Deshalb können die Regeln und Einstellungen nicht beim Kompilieren des Programms festgelegt werden, sondern sie müssen während des Laufes zugänglich sein.
- Notensatz ist eine Frage des Augenmaßes, und damit auch vom Geschmack abhängig. Benutzer können mit unseren Entscheidungen unzufrieden sein. Darum müssen also auch die Definitionen des typographischen Stils dem Benutzer zugänglich sein.
- Schließlich verfeinern wir unseren Formatierungsalgorithmus immer weiter, also müssen die Regeln auch flexibel sein. Die Sprache C++ zwingt zu einer bestimmten Gruppierungsmethode, die nicht den Regeln für den Notensatz entspricht.

Diese Probleme wurden angegangen, indem ein Übersetzer für die Programmiersprache Scheme integriert wurde und Teile von LilyPond in Scheme neu geschrieben wurden. Die derzeitige Formatierungsarchitektur ist um die Notation von graphischen Objekten herum aufgebaut, die von Scheme-Variablen und -Funktionen beschrieben werden. Diese Architektur umfasst Formatierungsregeln, typographische Stile und individuelle Formatierungsentscheidungen. Der Benutzer hat direkten Zugang zu den meisten dieser Einstellungen.

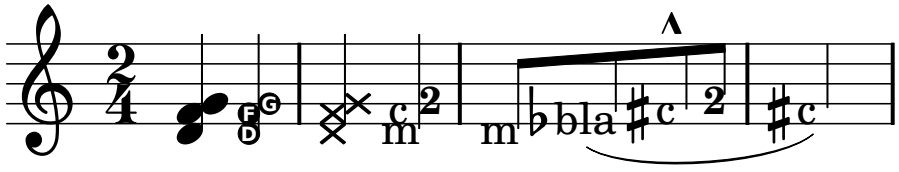
Scheme-Variablen steuern Layout-Entscheidungen. Zum Beispiel haben viele graphische Objekte eine Richtungsvariable, die zwischen oben und unten (oder rechts und links) wählen kann. Hier etwa sind zwei Akkorde mit Akzenten und Arpeggien. Beim ersten Akkord sind alle Objekte nach unten (oder links) ausgerichtet, beim zweiten nach oben (rechts).



Der Prozess des Notensetzens besteht für das Programm darin, die Variablen der graphischen Objekte zu lesen und zu schreiben. Einige Variablen haben festgelegte Werte. So ist etwa die Dicke von vielen Linien – ein Charakteristikum des typographischen Stils – von vornherein festgelegt. Wenn sie geändert werden, ergibt sich ein anderer typographischer Eindruck.



Formatierungsregeln sind auch vorbelegte Variablen. Zu jedem Objekt gehören Variablen, die Prozeduren enthalten. Diese Prozeduren machen die eigentliche Satzarbeit, und wenn man sie durch andere ersetzt, kann die Darstellung von Objekten verändert werden. Im nächsten Beispiel wird die Regel, nach der die Notenköpfe gezeichnet werden, während des Ausschnitts verändert.



Welche Symbole?

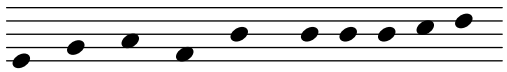
Während des Notensatzprozesses entscheidet sich, wo Symbole platziert werden. Das kann aber nur gelingen, wenn vorher entschieden wird, *welche* Symbole gesetzt werden sollen, also welche Notation benutzt werden soll.

Die heutige Notation ist ein System zur Musikaufzeichnung, das sich über die letzten 1000 Jahre entwickelt hat. Die Form, die heute üblicherweise benutzt wird, stammt aus dem frühen Barock. Auch wenn sich die grundlegenden Formen (also die Notenköpfe, das Fünfliniensystem) nicht verändert haben, entwickeln sich die Details trotzdem immer noch weiter, um die Errungenschaften der Neuen Musik darstellen zu können. Die Notation umfasst also 500 Jahre Musikgeschichte. Ihre Anwendung reicht von monophonen Melodien bis zu ungeheurem Kontrapunkt für großes Orchester.

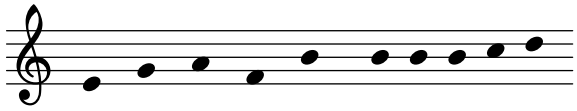
Wie bekommen wir dieses vielköpfige Monster zu fassen? Unsere Lösung ist es, eine strikte Trennung zwischen der Notation, also welche Symbole benutzt werden, und dem Satz, also wohin sie gesetzt werden, zu machen. Um das Problem anzupacken, haben wir es in kleine (programmierbare) Happen zerteilt, so dass jede Art von Symbol durch ein eigenes Plugin verarbeitet wird. Alle Plugins kooperieren durch die LilyPond-Architektur. Sie sind vollständig modular und unabhängig und können somit auch unabhängig voneinander entwickelt werden. Der Schreiber, der die Musik in Graphik umwandelt, ist ein Kopist oder Notenstecher (engl. engraver). Darum werden die Plugins als `engraver` bezeichnet.

Im nächsten Beispiel wird gezeigt, wie mit dem Plugin für die Notenköpfe, dem `Note_heads_engraver` („Notenkopfstecher“) der Satz begonnen wird.

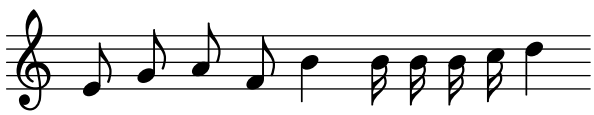
Dann fügt ein `Staff_symbol_engraver` („Notensystemstecher“) die Notenlinien hinzu.



Der `Clef_engraver` („Notenschlüsselstecher“) definiert eine Referenzstelle für das System.

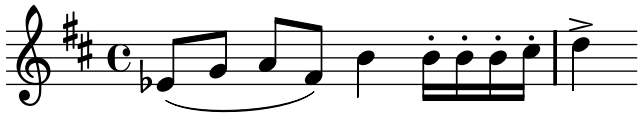


Der `Stem_engraver` („Halsstecher“) schließlich fügt Hälse hinzu.

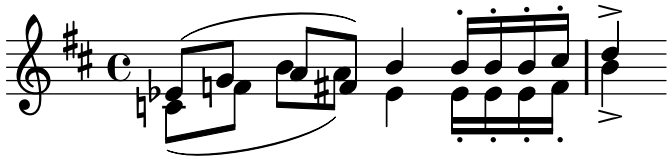


Dem `Stem_engraver` wird jeder Notenkopf mitgeteilt, der vorkommt. Jedes Mal, wenn ein Notenkopf erscheint (oder mehrere bei einem Akkord), wird ein Hals-Objekt erstellt und an den Kopf geheftet. Wenn wir dann noch engraver für Balken, Bögen, Akzente, Vorzeichen,

Taktlinien, Taktangaben und Tonartbezeichnungen hinzufügen, erhalten wir eine vollständige Notation.



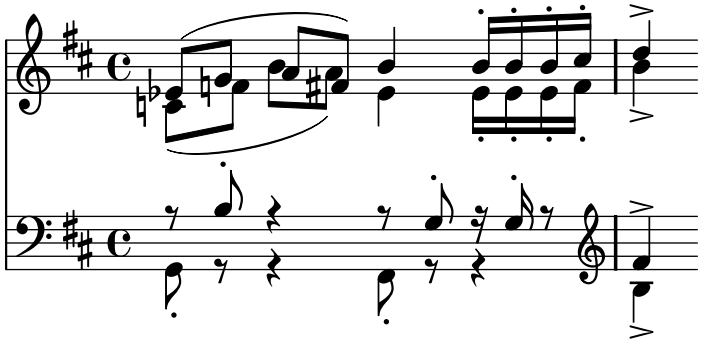
Dieses System funktioniert gut für monophone Musik, aber wie geht es mit Polyphonie? Hier müssen sich mehrere Stimmen ein System teilen.



In diesem Fall benutzen beide Stimmen das System und die Vorzeichen gemeinsam, aber die Hälse, Bögen, Balken usw. sind jeder einzelnen Stimme eigen. Die engraver müssen also gruppiert werden. Die Köpfe, Hälse, Bögen usw. werden in einer Gruppe mit dem Namen „Voice context“ (Stimmenkontext) zusammengefasst, die engraver für den Schlüssel, die Vorzeichen, Taktstriche usw. dagegen in einer Gruppe mit dem Namen „Staff context“ (Systemkontext). Im Falle von Polyphonie hat ein Staff-Kontext dann also mehr als nur einen Voice-Kontext. Auf gleiche Weise können auch mehrere Staff-Kontexte in einen großen Score-Kontext (Partiturkontext) eingebunden werden.

Siehe auch

Programmreferenz: Abschnitt “Contexts” in *Programmreferenz*.



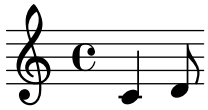
Die Darstellung der Musik

Idealerweise ist das Eingabeformat für ein höheres Satzsystem die abstrakte Beschreibung des Inhaltes. In diesem Fall wäre das die Musik selber. Das stellt uns aber vor ein ziemlich großes Problem, denn wie können wir definieren, was Musik wirklich ist? Anstatt darauf eine Antwort zu suchen, haben wir die Frage einfach umgedreht. Wir schreiben ein Programm, das den Notensatz beherrscht und passen das Format an, so einfach wie möglich zu sein. Wenn es nicht mehr vereinfacht werden kann, haben wir per Definition nur noch den reinen Inhalt. Unser Format dient als die formale Definition eines Musiktextes.

Die Syntax ist gleichzeitig die Benutzerschnittstelle bei LilyPond, darum soll sie einfach zu schreiben sein; z. B. bedeutet

```
c'4 d'8
```

eine Viertel c' und eine Achtel d', wie in diesem Beispiel:



In kleinem Rahmen ist diese Syntax sehr einfach zu benutzen. In größeren Zusammenhängen aber brauchen wir Struktur. Wie sonst kann man große Opern oder Symphonien notieren? Diese Struktur wird gewährleistet durch sog. music expressions (Musikausdrücke): indem kleine Teile zu größeren kombiniert werden, kann komplexere Musik dargestellt werden. So etwa hier:

```
c4
```

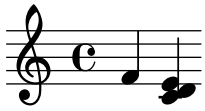
Gleichzeitig erklingende Noten werden hinzugefügt, indem man alle in << und >> einschließt.

```
<<c4 d4 e4>>
```

Um aufeinanderfolgende Noten darzustellen, werden sie in geschweifte Klammern gefasst:

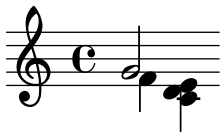
```
{ ... }
```

```
{ f4 <<c4 d4 e4>> }
```



Dieses Gebilde ist in sich wieder ein Ausdruck, und kann daher mit einem anderen Ausdruck kombiniert werden (hier mit einer Halben).

```
<< g2 \\ { f4 <<c4 d4 e4>> } >>
```

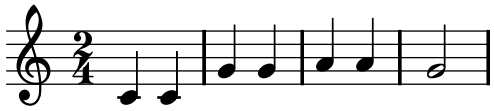


Solche geschachtelten Strukturen können sehr gut in einer kontextunabhängigen Grammatik beschrieben werden. Der Programmcode für den Satz ist auch mit solch einer Grammatik erstellt. Die Syntax von LilyPond ist also klar und ohne Zweideutigkeiten definiert.

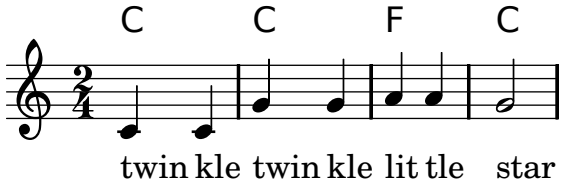
Die Benutzerschnittstelle und die Syntax werden als erstes vom Benutzer wahrgenommen. Teilweise sind sie eine Frage des Geschmackes und werden viel diskutiert. Auch wenn Geschmacksfragen ihre Berechtigung haben, sind sie nicht sehr produktiv. Im großen Rahmen von LilyPond spielt die Eingabe-Syntax nur eine geringe Rolle, denn eine logische Syntax zu schreiben ist einfach, guten Formatierungscode aber sehr viel schwieriger. Das kann auch die Zeilenzahl der Programmzeilen zeigen: Analysieren und Darstellen nimmt nur etwa 10% des Codes ein:

Beispielanwendung

Wir haben LilyPond als einen Versuch geschrieben, wie man die Kunst des Musiksatzes in ein Computerprogramm gießen kann. Dieses Programm kann nun dank vieler harter Arbeitsstunden benutzt werden, um sinnvolle Aufgaben zu erledigen. Die einfachste ist dabei der Notendruck.



Indem wir Akkordsymbole und einen Text hinzufügen, erhalten wir ein Lead Sheet.

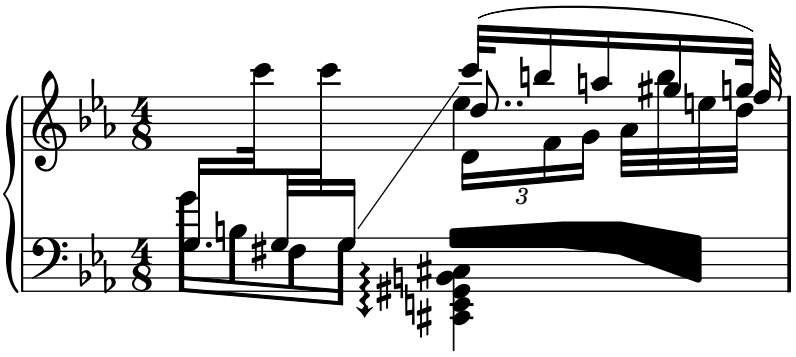


Mehrstimmige Notation und Klaviermusik kann auch gesetzt werden. Das nächste Beispiel zeigt einige etwas exotischere Konstruktionen:

Screech and boink

Random complex notation

Han-Wen Nienhuys



Die obenstehenden Beispiele wurde manuell erstellt, aber das ist nicht die einzige Möglichkeit. Da der Satz fast vollständig automatisch abläuft, kann er auch von anderen Programmen angesteuert werden, die Musik oder Noten verarbeiten. So können etwa ganze Datenbanken musikalischer Fragmente automatisch in Notenbilder umgewandelt werden, die dann auf Internetseiten oder in Multimediapräsentation Anwendung finden.

Dieses Benutzerhandbuch zeigt eine weitere Möglichkeit: Die Noten werden als reiner Text eingegeben und können darum sehr einfach integriert werden in andere textbasierte Formate wie etwa LaTeX, HTML oder, wie in diesem Fall, Texinfo. Durch ein spezielles Programm werden die Eingabefragmente durch Notenbilder in der resultierenden PDF- oder HTML-Datei ersetzt. Dadurch ist es sehr einfach, Noten und Text zu kombinieren.

1.2 Über die Dokumentation

Die Dokumentation zu LilyPond ist unterteilt in mehrere Handbücher.

Über das Handbuch zum Lernen (LM)

Dieses Handbuch erklärt die Grundbegriffe von LilyPond und stellt die fundamentalen Konzepte hinter dem Programm vor. Diese Kapitel sollten in linearer Reihenfolge gelesen werden.

Am Ende jedes Abschnitts findet sich ein Absatz **Siehe auch**, der Kreuzreferenzen zu anderen Abschnitten enthält. Beim ersten Durchlesen empfiehlt es sich nicht, diesen gleich zu folgen, da meist noch zahlreiche Grundbegriffe zum Verständnis fehlen. Wenn Sie sich durch das Handbuch zum Lernen geackert haben, wollen Sie vielleicht einzelne Abschnitte nochmal durchgehen und dann den Kreuzverweisen zur Vertiefung der Zusammenhänge folgen.

- Kapitel 1 [Einleitung], Seite 2: erklärt den Hintergrund und das Ziel von LilyPond.
- Kapitel 2 [Übung], Seite 12: liefert eine einfache Einführung in das Setzen von Musik mit LilyPond. Neulinge sollten mit diesem Kapitel beginnen.
- Kapitel 3 [Grundbegriffe], Seite 41: erklärt etliche allgemeine Konzepte hinter dem Dateiformat von LilyPond. Wenn Sie sich nicht sicher sind, an welcher Stelle Sie einen Befehl in die Datei einfügen sollen, ist dieses Kapitel genau das richtige!
- Kapitel 4 [Die Ausgabe verändern], Seite 68: stellt dar, wie die Standardeinstellungen von LilyPond verändert werden können.
- Kapitel 5 [An LilyPond-Projekten arbeiten], Seite 76: liefert Tipps im praktischen Umgang mit Lilypond und gibt Hinweise, wie gängige Fehler vermieden werden können. Bevor Sie mit einem großen Projekt beginnen, sollten Sie dieses Kapitel unbedingt gelesen haben!

Das Handbuch zum Lernen enthält auch zahlreiche Anhänge, die nicht zum linearen Durchlesen geeignet sind. Sie sind allerdings zur späteren Referenz sehr gut geeignet:

- Anhang A [Vorlagen], Seite 87: zeigt einige fertige Dokumentvorlagen für diverse Stücke mit unterschiedlichen Charakteristika. Kopieren Sie einfach die Vorlagen in Ihre eigene Datei, fügen Sie die Noten hinzu und Sie sind fertig!
- Anhang B [Scheme-Übung], Seite 118: liefert eine kurze Einführung in Scheme, die Programmiersprache, die die Musikfunktionen in LilyPond intern benutzen. Dies stellt tiefgehendes Wissen dar, wenn Sie LilyPond bis ins kleinste Detail konfigurieren möchten. Die meisten Benutzer brauchen dies jedoch selten bis gar nicht.

Über das Glossar (MG)

Abschnitt “Das Musikglossar” in *Glossar* erklärt musikalische Fachausdrücke und enthält auch deren Übersetzungen in diverse Sprachen. Wenn Sie mit Musiknotation oder der (englischsprachigen) Musikterminologie nicht vertraut sind (vor allem, wenn Englisch nicht Ihre Muttersprache ist), ist es sehr empfehlenswert, das Musikglossar immer wieder zu Rate zu ziehen.

Über die Notationsreferenz (NR)

In diesem Buch werden alle LilyPond-Befehle erklärt, die Notationszeichen produzieren. Es geht von der Annahme aus, dass der Leser sich mit den Grundkonzepten des Programmes im Handbuch zum Lernen bekannt gemacht hat.

- *Abschnitt “Musikalische Notation” in Benutzerhandbuch*: erklärt alles über die grundlegenden Notationskonstruktionen. Dieses Kapitel ist für fast jedes Notationsprojekt nützlich.
- *Abschnitt “Spezielle Notation” in Benutzerhandbuch*: erklärt spezifische Schwierigkeiten, die sich bei bestimmten Notationstypen ergeben. Dieses Kapitel ist nur in entsprechenden Fällen bestimmter Instrumente oder bei Gesang zu konsultieren.

- *Abschnitt "Allgemeine Eingabe und Ausgabe" in Benutzerhandbuch*: erläutert allgemeine Informationen über die Eingabedateien von Lilypond und wie die Ausgabe gesteuert werden kann.
- *Abschnitt "Abstände" in Benutzerhandbuch*: befasst sich mit globalen Fragen wie der Definition von Papierformaten oder wie man Seitenumbrüche definiert.
- *Abschnitt "Standardeinstellungen verändern" in Benutzerhandbuch*: erklärt, wie des Layout getrimmt werden kann um genau zum gewünschten Ergebnis zu kommen.
- *Abschnitt "Schnittstellen für Programmierer" in Benutzerhandbuch*: demonstriert die Erstellung von musikalischen Funktionen.

Ds Benutzerhandbuch enthält auch Anhänge mit nützlichen Referenztabellen.

- Die Abschnitt "Literatur" in *Benutzerhandbuch* enthält einige wichtige Quellen für alle, die mehr über Notation und den Notensatz erfahren wollen.
- Abschnitt "Notationsübersicht" in *Benutzerhandbuch* sind Tabellen, in denen Akkordbezeichnungen, MIDI-Instrumente, Farbbezeichnungen und die Zeichen der Feta-Schriftart gesammelt sind.
- Die *Abschnitt "Befehlsübersicht" in Benutzerhandbuch* zeigt die wichtigsten LilyPond-Befehle.
- Der *Abschnitt "Index der LilyPond-Befehle" in Benutzerhandbuch* listet alle Befehle auf, die mit \ anfangen.
- Der *Anhang D [LilyPond-Index], Seite 127* ist ein vollständiger Index.

Über die Anwendungsbenutzung (AU)

In diesem Buch wird erklärt, wie das Programm ausgeführt wird und wie die Notation von LilyPond in andere Programme integriert werden kann.

- *Abschnitt "Installieren" in Programmbenutzung*: erklärt wie LilyPond installiert wird (inklusive Kompilation, wenn es nötig sein sollte).
- *Abschnitt "Setup" in Programmbenutzung*: erklärt wie der Computer eingerichtet wird, damit LilyPond optimal genutzt werden kann. Hierzu gehören etwa spezielle Umgebungen für bestimmte Texteditoren.
- *Abschnitt "LilyPond starten" in Programmbenutzung*: zeigt, wie LilyPond und seine Hilfsprogramme gestartet werden. Zusätzlich wird hier erklärt, wie Quelldateien von alten LilyPond-Versionen aktualisiert werden können.
- *Abschnitt "LilyPond-book" in Programmbenutzung*: erklärt die Details, um einen Text mit eingefügten Notenbeispielen (wie etwa dieses Handbuch) zu erstellen.
- *Abschnitt "Von anderen Formaten konvertieren" in Programmbenutzung*: erklärt, wie die Konvertierungsprogramme aufgerufen werden. Diese Programme kommen mit LilyPond zusammen und konvertieren eine Vielzahl von Notensatzformaten in das `.ly`-Format.

Über die Schnipselliste

Die Abschnitt "Schnipsel" in *Beispiele* sind eine ausführliche Sammlung kurzer Beispiele, anhand derer Tricks, Tipps und Spezialfunktionen von LilyPond demonstriert werden. Die meisten dieser Schnipsel können auch im LilyPond Schnipsel Depot betrachtet werden. Diese Internetseite verfügt auch über ein durchsuchbares LilyPond-Handbuch.

Die Liste der Schnipsel zu einem Abschnitt des Benutzerhandbuchs ist auch dort jeweils im Abschnitt **Siehe auch** verlinkt.

Über die Referenz der Interna (IR)

Die Abschnitt "Programmreferenz" in *Programmreferenz* ist eine Sammlung intensiv verlinkter HTML-Seiten, die alle Details jeder einzelnen LilyPond-Klasse, jedes Objektes und jeder Funktion erklären. Sie wird direkt aus den Satzdefinitionen im Quellcode produziert.

So gut wie alle Formatierungsmöglichkeiten, die intern verwendet werden, sind auch direkt für den Benutzer zugänglich. Alle Variablen z. B., die Dicke-Werte, Entfernungen usw. kontrollieren, können in den Eingabe-Dateien verändert werden. Es gibt eine riesige Anzahl von Formatierungsoptionen, und alle haben einen „Siehe"-Abschnitt, der auf die Dokumentation verweist. Im HTML-Handbuch haben diese Abschnitte klickbare Links.

Die Programmreferenz ist nur auf englisch verfügbar.

Andere Dokumentation

Es gibt noch eine Reihe weiterer wertvoller Informationsquellen zu LilyPond:

- Neuigkeiten: eine Zusammenfassung der Änderungen in LilyPond seit der letzten Version.
- Das Archiv der lilypond-user Mailing-Liste: enthält alle bisher an die Liste gesendeten Mails. Viele Fragen werden immer wieder gestellt und auch beantwortet. Die Chance, dass Ihre Frage auch schon mal aufgetaucht ist, ist also relativ groß. In diesem Fall finden Sie die Antwort in diesem Archiv.
- Das Archiv der lilypond-devel Mailing-Liste: enthält alle bisher an die Entwicklerliste gesendeten Mails. Diese Diskussionen sind dementsprechend technisch gehalten. Wenn Sie eine tiefergehende Frage zu den Interna von LilyPond haben, finden Sie die Antwort vielleicht in diesem Archiv.
- Eingebettete Musikbeispiele: Auf allen HTML-Seiten mit Notenbeispielen, die von LilyPond erzeugt wurden, kann die originale Quelldatei durch einen Klick auf das Bild betrachtet werden.
- Initialisierungsdateien von LilyPond:

  Der Speicherort der Dokumentationsdateien unterscheidet sich evtl. je nach Betriebssystem. Manchmal wird hier auf Initialisierungs- oder Beispieldateien verwiesen. Das Handbuch nimmt dabei an, dass diese Dateien sich relativ zum Quellverzeichnis befinden. Zum Beispiel würde der Pfad '`input/lsr/Verzeichnis/bla.ly`' etwa auf die Datei '`lilypond2.x.y/input/lsr/Verzeichnis/bla.ly`' verweisen. In den Binärpaketen für Unix-Plattformen sind Dokumentation und Beispiele üblicherweise in einem Verzeichnis wie '`/usr/share/doc/lilypond/`' gespeichert. Initialisierungsdateien, etwa '`scm/lily.scm`' oder '`ly/engraver-init.ly`', befinden sich normalerweise im Verzeichnis '`/usr/share/lilypond/`'.

  Weiterführende Informationen finden Sie unter Abschnitt 4.7.3 [Mehr Information], Seite 74.

2 Übung

Diese Übung führt ein in die Notationssprache des Programmes LilyPond und erklärt, wie man damit Noten setzen kann. Nach einer ersten Einleitung wird erklärt, wie die häufigsten Notenbilder in schönen Notendruck umgesetzt werden können.

2.1 Erste Schritte

In diesem Abschnitt werden die Grundlagen zur Benutzung des Programmes erklärt.

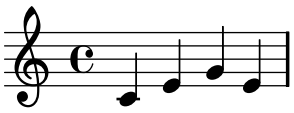
2.1.1 Eine Quelldatei übersetzen

„Kompilation“ ist der Begriff, der benutzt wird, um eine Lilypond-Eingabedatei mit dem Programm LilyPond in eine Notenausgabe umzuwandeln, die ausgedruckt werden kann. Zusätzlich besteht die Option, eine MIDI-Datei zu produzieren, die abgespielt werden kann. Das erste Beispiel zeigt, wie solch eine einfache Eingabedatei ausschauen kann.

Um Notensatz zu erstellen, muss die Notation in der Eingabedatei beschrieben werden. Wenn man z.B. schreibt:

```
{
  c' e' g' e'
}
```

so erhält man folgendes Resultat:



Achtung: In jeder LilyPond-Datei müssen `{` **geschweifte Klammern** `}` um die Noten oder Gesangstext gesetzt werden. Vor und hinter den Klammern sollten Leerzeichen eingegeben werden, damit keine Unklarheiten in Verbindung mit den eigentlichen Notensymbolen entstehen. An Anfang und Ende der Zeile können diese Leerzeichen auch weggelassen werden. Es kann sein, dass in diesem Handbuch die Klammern in manchen Beispielen fehlen, aber man sollte immer daran denken, sie in den eigenen Dateien zu benutzen! Mehr Informationen zu der Darstellung der Beispiele in diesem Handbuch gibt der Abschnitt Abschnitt 2.1.4 [Wie soll das Handbuch gelesen werden], Seite 19.

Zusätzlich unterscheidet LilyPond **Groß- und Kleinschreibung**. `{ c d e }` ist zulässiger Code, `{ C D E }` dagegen resultiert in einer Fehlermeldung.

Eingabe von Noten und Ansicht des Ergebnisses

In diesem Kapitel zeigen wir, welche Kommandos eingegeben werden müssen, um ein Notenbild zu erzeugen, und wie das Resultat dann betrachtet werden kann.

Beachten Sie, dass es eine Reihe an Texteditoren mit besserer Unterstützung für LilyPond gibt. Mehr dazu im Abschnitt Abschnitt “Unterstützung von Texteditoren” in *Programmbenutzung*.

Achtung: Das erste Mal, wenn Sie LilyPond benutzen, kann es eine Minute oder länger dauern, weil das Programm zuerst alle Schriftarten, die auf dem System zur Verfügung stehen, untersucht. Aber nach diesem ersten Mal läuft LilyPond sehr viel schneller.

MacOS X

Wenn Sie das `LilyPond.app`-Symbol doppelt klicken, öffnet sich eine Beispiel-Datei. Speichern Sie sie etwa als '`test.ly`' auf dem Desktop und übersetzen Sie sie mit dem Menü-Befehl „Compile > Typeset File“. Die PDF-Datei mit dem fertigen Notensatz wird automatisch geöffnet.

Das nächste Mal, wenn Sie LilyPond benutzen, sollten Sie „New“ oder „Open“ wählen. Sie müssen die Datei speichern, bevor Sie sie übersetzen können. Wenn es Fehler gibt, lesen Sie die Meldungen im Log-Fenster.

Windows

Wenn sie auf das LilyPond-Symbol auf dem Desktop doppelklicken, öffnet sich ein einfacher Texteditor mit einer Beispieldatei. Speichern Sie sie z. B. als '`test.ly`' auf dem Desktop und klicken Sie dann doppelt auf die Datei, um die Übersetzung zu beginnen (das Dateisymbol ist eine Note). Nach einigen Sekunden wird eine Datei '`test.pdf`' auf dem Desktop erscheinen. Mit einem Doppelklick kann das fertige Notenbild in der PDF-Datei angezeigt werden. Eine Alternative ist es, die '`test.ly`'-Datei mit der Maus auf das LilyPond-Symbol zu ziehen.

Um eine schon existierende Datei zu bearbeiten, klicken Sie mit der rechten Maustaste auf die Datei und wählen Sie „Edit source“. Um eine leere Datei zu erhalten, mit der Sie ein neues Stück beginnen können, öffnen Sie den Texteditor durch Doppelklick auf das LilyPond-Symbol und benutzen Sie „New“ im „File“-Menü, oder klicken Sie mit der rechten Maustaste auf den Desktop und wählen Sie „Neu...Textdatei“, ändern Sie dann den Namen so wie Sie möchten und ändern Sie die Dateiendung in `.ly`. Doppelklicken Sie auf die Datei, um Ihren LilyPond-Eingabecode einzugeben.

Mit dem Doppelklick wird nicht nur die PDF-Datei erstellt, sondern auch eine '`.log`'-Datei. Hier wird gespeichert, was LilyPond aus der Quelldatei gelesen hat. Sollten Fehler auftreten, hilft oft ein Blick in diese Datei.

UNIX

Erstellen Sie eine Text-Datei mit dem Namen '`test.ly`' und geben Sie folgenden Text ein:

```
{
  c' e' g' e'
}
```

Um die Datei zu bearbeiten, geben sie an der Konsole

```
lilypond test.ly
```

ein. Sie werden ungefähr folgende Meldungen sehen:

```
lilypond test.ly
GNU LilyPond 2.11.58

»test.ly« wird verarbeitet
Analysieren...
Interpretation der Musik...
Vorverarbeitung der grafischen Elemente...
Ideale Seitenanzahl wird gefunden...
Musik wird auf eine Seite angepasst...
Systeme erstellen...
```

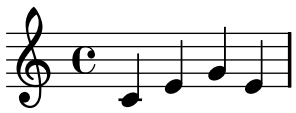
```
Layout nach »test.ps« ausgeben...
Konvertierung nach »test.pdf«...
```

Als Ergebnis erhalten Sie ein '`test.pdf`', das Sie mit den Standardprogrammen Ihres Betriebssystemes anschauen können.

2.1.2 Einfache Notation

LilyPond fügt einige Bestandteile des Notenbildes automatisch hinzu. Im nächsten Beispiel sind nur vier Tonhöhen angegeben, aber LilyPond setzt trotzdem einen Schlüssel, eine Taktangabe und Notendauern.

```
{
  c' e' g' e'
}
```



Diese Einstellungen können verändert werden, aber in den meisten Fällen sind die automatischen Werte durchaus brauchbar.

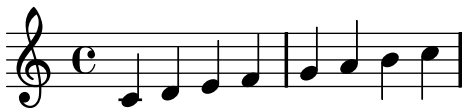
Tonhöhen

Glossar: Abschnitt "Tonhöhe" in *Glossar*, Abschnitt "Intervalle" in *Glossar*, Abschnitt "Tonleiter" in *Glossar*, Abschnitt "eingestrichenes C" in *Glossar*, Abschnitt "Oktave" in *Glossar*, Abschnitt "Versetzungszeichen" in *Glossar*.

Die Tonhöhen werden mit Kleinbuchstaben eingegeben, die den Notennamen entsprechen. Es ist jedoch wichtig zu wissen, dass LilyPond in seiner Standardeinstellung die englischen Notennamen verwendet. Bis auf eine Ausnahme entsprechen sie den deutschen, deshalb wird die Voreinstellung von LilyPond für diese Übung beibehalten. Die *Ausnahme* ist das h – in LilyPond muss man anstelle dessen b schreiben! Das deutsche b dagegen wird als bes notiert, ein his dagegen würde bis geschrieben. Siehe auch Abschnitt "Versetzungszeichen" in *Benutzerhandbuch* und Abschnitt "Notenbezeichnungen in anderen Sprachen" in *Benutzerhandbuch*, hier wird beschrieben, wie sich die deutschen Notennamen benutzen lassen.

Am einfachsten können Noten im `\relative`-Modus eingegeben werden. In diesem Modus wird die Oktave der Note automatisch gewählt, indem angenommen wird, dass die folgende Note immer so nah wie möglich in Bezug auf die vorhergehende gesetzt wird, d. h. sie wird höchstens drei Notenzeilen höher oder tiefer als die vorhergehende Note gesetzt. Fangen wir unser erstes Notationsbeispiel mit einer *scale* an, wo also die nächste Note immer nur eine Notenlinie über der vorherigen steht.

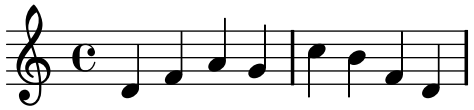
```
% Beginnpunkt auf das mittlere C setzen
\relative c' {
  c d e f
  g a b c
}
```



Die erste Note ist ein *eingestrichenes C*. Jede folgende Note befindet sich so nah wie möglich bei der vorherigen – das erste ‚C' ist also das nächste C vom eingestrichenen C aus gerechnet.

Darauf folgt das nächstmögliche D in Bezug auf die vorhergehende Note. Mit diesen Regeln können auch Melodien mit größeren Intervallen im `\relative`-Modus gebildet werden:

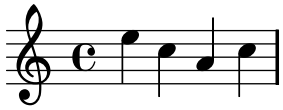
```
\relative c' {
  d f a g
  c b f d
}
```



Es ist nicht notwendig, dass die erste Note der Melodie mit der Note beginnt, die die erste Tonhöhe angibt. Die erste Note (das ‚D') des vorigen Beispiels ist das nächste D vom eingestrichenen C aus gerechnet.

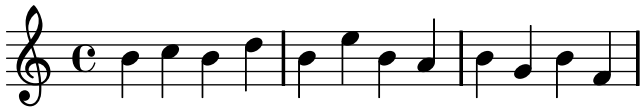
Indem man Apostrophe `'` (Taste Shift+#) oder Kommata `,` zu dem `\relative c' {` hinzufügt oder entfernt, kann die Oktave der ersten Tonhöhe verändert werden:

```
% zweigestrichenes C
\relative c'' {
  e c a c
}
```



Der relative Modus kann zunächst verwirrend erscheinen, aber es ist die einfachste Art, die meisten Melodien zu notieren. Schauen wir uns an, wie diese relative Berechnung in der Praxis funktioniert. Wenn wir mit einem H beginnen (`b` in der LilyPond-Syntax), welches sich auf der mittleren Linie im Violinschlüssel befindet, können wir C, D und E aufwärts notieren, und A, G und F unter dem H. Wenn also die Note, die auf das H folgt, ein C, D oder E ist, setzt LilyPond es oberhalb des Hs, wenn es ein A, G oder F ist, wird es darunter gesetzt.

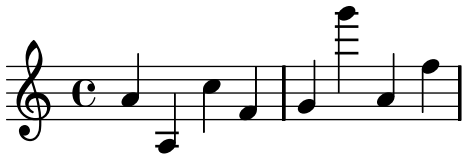
```
\relative c'' {
  b c  % c ist 1 Zeile aufwärts, also c über dem b
  b d  % d ist 2 Zeilen aufwärts, oder 5 runter, also d über dem b
  b e  % e ist 3 aufwärts oder 4 runter, also e über dem b
  b a  % a ist 6 aufwärts oder 1 runter, also a unter dem b
  b g  % g ist 5 aufwärts oder 2 runter, also g unter dem b
  b f  % f ist 4 aufwärts oder 3 runter, also f unter dem b
}
```



Die gleiche Berechnung findet auch statt, wenn eine der Noten erhöht oder erniedrigt ist. *Versetzungszeichen* werden **vollständig ignoriert** bei der Berechnung. Genau die gleiche Berechnung wird analog von jeder folgenden Tonhöhe aus für die nächste Tonhöhe neu ausgeführt.

Um Intervalle zu notieren, die größer als drei Notenzeilen sind, kann man die Oktave verändern. Mit einem Apostroph `'` (Taste Shift+#) direkt hinter dem Notennamen wird die Oktave um eins erhöht, mit einem Komma `,` um eins erniedrigt.

```
\relative c'' {
  a a, c' f,
  g g'' a,, f'
}
```



Um eine Notenhöhe um zwei (oder mehr!) Oktaven zu verändern, werden sukzessive `''` oder `,,` benutzt – es muss sich dabei wirklich um zwei einzelne Apostrophen und nicht um das Anführungszeichen `"` (Taste Shift+2) handeln! Auch die Anfangsoktave für einen `\relative c'`-Abschnitt kann so verändert werden.

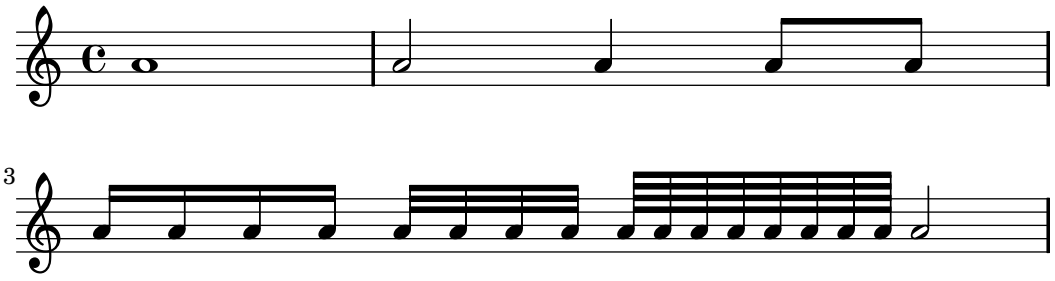
Tondauern (Rhythmen)

Glossar: Abschnitt "Balken" in *Glossar*, Abschnitt "Tondauer" in *Glossar*, Abschnitt "ganze Note" in *Glossar*, Abschnitt "halbe Note" in *Glossar*, Abschnitt "Viertelnote" in *Glossar*, Abschnitt "punktierte Note" in *Glossar*.

Die *Dauer* einer Note wird durch eine Zahl bezeichnet, die direkt auf den Notennamen folgend eingegeben wird. `1` für eine *ganze Note*, `2` für eine *halbe Note*, `4` für eine *Viertelnote* und so weiter. *Notenhälse* und *Balken* werden automatisch hinzugefügt.

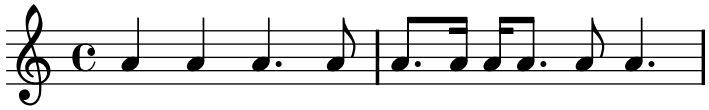
Wenn keine Dauer bezeichnet wird, wird die der vorhergehenden Note verwendet. Für die erste Note ist eine Viertel als Standard definiert.

```
\relative c'' {
  a1
  a2 a4 a8 a
  a16 a a a a32 a a a a64 a a a a a a a2
}
```



Um *punktierte Noten* zu erzeugen, wird einfach ein Punkt `.` hinter die Notendauer geschrieben. Die Dauer einer punktierten Note muss explizit, also inklusive der Nummer, angegeben werden.

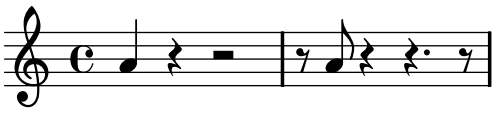
```
\relative c'' {
  a a a4. a8
  a8. a16 a a8. a8 a4.
}
```



Pausen

Eine *Pause* wird genauso wie eine Noten eingegeben; ihre Bezeichnung ist `r` :

```
\relative c'' {
  a r r2
  r8 a r4 r4. r8
}
```

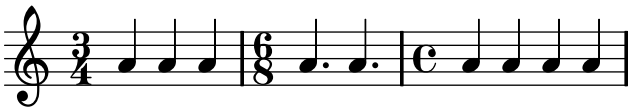


Taktangabe

Glossar: Abschnitt “Taktangabe” in *Glossar*.

Die *Taktart* kann mit dem `\time`-Befehl definiert werden:

```
\relative c'' {
  \time 3/4
  a4 a a
  \time 6/8
  a4. a
  \time 4/4
  a4 a a a
}
```

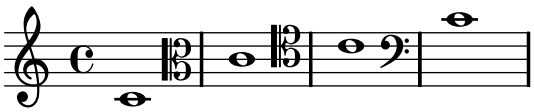


Notenschlüssel

Glossar: Abschnitt “Taktangabe” in *Glossar*.

Der *Notenschlüssel* kann mit dem `\clef`-Befehl gesetzt werden:

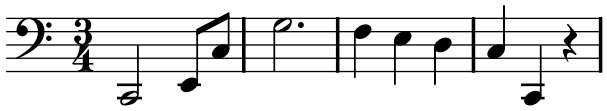
```
\relative c' {
  \clef treble
  c1
  \clef alto
  c1
  \clef tenor
  c1
  \clef bass
  c1
}
```



Alles zusammen

Hier ist ein kleines Beispiel, dass all diese Definitionen beinhaltet:

```
\relative c, {
  \time 3/4
  \clef bass
  c2 e8 c' g'2.
  f4 e d c4 c, r4
}
```



Siehe auch

Benutzerhandbuch: Abschnitt “Tonhöhen setzen” in *Benutzerhandbuch*, Abschnitt “Rhythmen eingeben” in *Benutzerhandbuch*, Abschnitt “Pausen eingeben” in *Benutzerhandbuch*, Abschnitt “Taktangabe” in *Benutzerhandbuch*, Abschnitt “Notenschlüssel” in *Benutzerhandbuch*.

2.1.3 Arbeiten an Eingabe-Dateien

LilyPonds Quelldateien ähneln Dateien in den meisten Programmiersprachen: Es ist auf Groß- und Kleinschreibung zu achten und Leerzeichen werden ignoriert. Ausdrücke werden mit geschweiften Klammern `{ }` eingeklammert und Kommentare mit dem Prozentzeichen `%` auskommentiert oder mit `%{ ... %}` umgeben.

Wenn das jetzt unverständlich erscheint, sind hier die Erklärungen:

- **Groß- und Kleinschreibung:** Die Bedeutung eines Zeichens verändert sich, je nachdem, ob es groß (`A, B, S, T`) oder klein (`a, b, s, t`) geschrieben wird. Noten müssen immer klein geschrieben werden, `{ c d e }` funktioniert, während `{ C D E }` einen Fehler produziert.
- **Leerzeichen:** Es spielt keine Rolle, wie viele Leerzeichen oder leere Zeilen sich zwischen den Zeichen der Quelldatei befinden. `{ c d e }` bedeutet das Gleiche wie `{ c  d e }` oder

```
{
c                    d
  e }
```

  Natürlich ist das letzte Beispiel etwas schwer zu lesen. Eine gute Daumenregel ist es, Code-Blöcke mit der Tab-Taste oder zwei Leerzeichen einzurücken:

```
{
  c d e
}
```

- **Ausdrücke:** Auch der kleinste Abschnitt an LilyPond-Code muss in `{` **geschweifte Klammern** `}` eingeschlossen werden. Diese Klammern zeigen LilyPond an, dass es sich um einen zusammengehörenden musikalischen Ausdruck handelt, genauso wie Klammern ‚()‘ in der Mathematik. Die Klammern sollten von jeweils einem Leerzeichen umgeben sein, um Zweideutigkeiten auszuschließen, es sei denn, sie befinden sich am Anfang oder Ende einer Zeile. Ein LilyPond-Befehl gefolgt von einem einfachen Ausdruck in Klammern (wie etwa `\relative { }`) wird auch als ein einzelner Musikausdruck gewertet.
- **Kommentare:** Ein Kommentar ist eine Bemerkung für den menschlichen Leser einer Quelldatei, es wird bei der Dateianalyse durch das Programm ignoriert, so dass es also keine Auswirkung auf die Druckausgabe der Noten hat. Es gibt zwei verschiedene Typen von Kommentaren. Das Prozentzeichen ‚%‘ geht einem Zeilen-Kommentar voraus: Alles nach diesem Zeichen wird in dieser Zeile ignoriert. Üblicherweise wird ein Kommentar *über* dem Code gesetzt, auf den es sich bezieht.

```
a4 a a a
% Dieser Kommentar bezieht sich auf das H
b2 b
```

Ein Block-Kommentar ist ein ganzer Abschnitt mit einem Kommentar. Alles, was von `%{` und `%}` umgeben ist, wird ignoriert. Das heißt, dass sich ein Block-Kommentar nicht ein einem anderen Blockkommentar befinden kann. Wenn Sie das versuchen sollten, beendet schon das erste `%}` *beide* Block-Kommentare. Das folgende Beispiel zeigt eine mögliche Anwendung von Kommentaren:

```
% Noten für twinkle twinkle hier
  c4 c g' g a a g2

%{
    Diese Zeilen, und die Noten unten werden
    ignoriert, weil sie sich in einem Block-Kommentar
    befinden.

    f f e e d d c2
%}
```

2.1.4 Wie soll das Handbuch gelesen werden

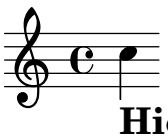
LilyPond-Code muss immer von `{ }` Zeichen oder einem `\relative c'' { ... }` umgeben sein, wie gezeigt in Abschnitt 2.1.3 [Arbeiten an Eingabe-Dateien], Seite 18. Im Rest dieses Handbuchs werden die meisten Beispiel allerdings darauf verzichten. Um sie zu reproduzieren, können Sie den entsprechenden Quellcode kopieren und in eine Textdatei einfügen, aber Sie **müssen** dabei `\relative c'' { }` einfügen, wie hier gezeigt:

```
\relative c'' {
  ... hier das Beispiel ...
}
```

Warum werden die Klammern hier meist weggelassen? Die meisten der Beispiele können in ein längeres Musikstück hineinkopiert werden, und dann ist es natürlich nicht sinnvoll, wenn auch noch `\relative c'' { }` dazukommt; ein `\relative` darf nicht innerhalb eines anderen `\relative` gesetzt werden, deshalb wird es hier weggelassen, damit die Beispiele auch innerhalb eines anderen Kontextes funktionieren. Wenn bei jedem Beispiel `\relative c'' { }` eingesetzt würde, könnten Sie die kleinen Beispiele der Dokumentation nicht einfach zu Ihrem eigenen Notentext hinzufügen. Die meisten Benutzer wollen Noten zu einer schon bestehenden Datei irgendwo in der Mitte hinzufügen, deshalb wurde der relative Modus für die Beispiele im Handbuch weggelassen.

Anklickbare Beispiele

Viele Leute lernen Programme, indem sie einfach herumprobieren. Das geht auch mit LilyPond. Wenn Sie in der HTML-Version dieses Handbuchs eine Abbildung anklicken, erhalten sie exakt den LilyPond-Code, der zum Satz der Abbildung benutzt wurde. Versuchen Sie es mit dieser Abbildung:



Hier klicken!

Wenn Sie einfach alles kopieren, was im „ly snippet“-Abschnitt steht, und in eine Text-Datei einfügen, haben Sie schon eine fertige Vorlage für weitere Experimente. Damit Sie genau das

gleiche Erscheinungsbild wie bei dem Beispiel selber erreichen, müssen Sie alles kopieren ab der Zeile „Start cut-&-pastable section“ bis ganz zum Ende der Datei.

Siehe auch

Mehr Hinweise dazu, wie LilyPond-Eingabedateien konstruiert werden sollten, finden sich in Abschnitt 5.1 [Vorschläge], Seite 76. Es ist aber wahrscheinlich am Besten, zuerst die gesamte Übung zu lesen.

2.2 Notation auf einem System

Dieses Kapitel lehrt grundlegende Bestandteile der Notation, die für eine Stimme auf einem System gebraucht werden.

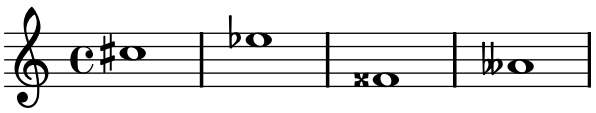
2.2.1 Versetzungszeichen und Tonartbezeichnung (Vorzeichen)

Versetzungszeichen

Glossar: Abschnitt “Kreuz” in *Glossar*, Abschnitt “B” in *Glossar*, Abschnitt “Doppelkreuz” in *Glossar*, Abschnitt “Doppel-B” in *Glossar*, Abschnitt “Versetzungszeichen” in *Glossar*.

Ein *Kreuz*-Versetzungszeichen[1] wird eingegeben, indem an den Notennamen ein ‚is‘ gehängt wird, ein *B*-Versetzungszeichen durch Anhängen von ‚es‘. Logischerweise wird dann ein *Doppelkreuz* oder *Doppel-B* durch Anhängen von ‚isis‘ oder ‚eses‘ geschrieben. Diese Syntax stammt aus der Tradition der germanischen Sprachen und ist also für deutsche Benutzer kein Problem. Es ist aber möglich, die Namen für die *Versetzungszeichen* in anderen Sprachen zu benutzen, siehe Abschnitt “Notenbezeichnungen in anderen Sprachen” in *Benutzerhandbuch*.

```
cis1 ees fisis, aeses
```

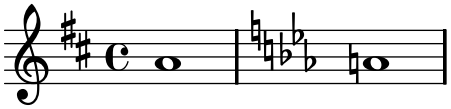


Tonartbezeichnungen (Vorzeichen)

Glossar: Abschnitt “Tonartbezeichnung” in *Glossar*, Abschnitt “Dur” in *Glossar*, Abschnitt “Moll” in *Glossar*.

Die *Tonart* eines Stückes wird erstellt mit dem Befehl `\key`, gefolgt von einer Notenbezeichnung und `\major` (für Dur) oder `\minor` (für Moll).

```
\key d \major
a1
\key c \minor
a
```



[1] In der Umgangssprache werden die Versetzungszeichen häufig auch Vorzeichen genannt. In diesem Handbuch wird jedoch zwischen Vorzeichen zur generellen Angabe der Tonart und den Versetzungszeichen, die direkt im Notentext erscheinen, unterschieden.

Warnung: Tonartbezeichnungen und Tonhöhen

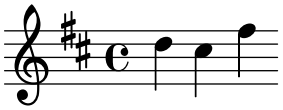
Glossar: Abschnitt “Versetzungszeichen” in *Glossar*, Abschnitt “Tonartbezeichnung” in *Glossar*, Abschnitt “Tonhöhe” in *Glossar*, Abschnitt “B” in *Glossar*, Abschnitt “Auflösungszeichen” in *Glossar*, Abschnitt “Kreuz” in *Glossar*, Abschnitt “Transposition” in *Glossar*.

Um zu bestimmen, ob vor einer bestimmten Note ein Versetzungszeichen erscheinen soll, untersucht LilyPond die Notenhöhen und die Tonart. Die Tonart beeinflusst nur die *gedruckten* Versetzungszeichen, nicht die wirklichen Tonhöhen! Diese Besonderheit scheint am Anfang oft verwirrend, so dass sie hier etwas genauer betrachtet wird.

LilyPond unterscheidet strickt zwischen dem musikalischen Inhalt und dem Satz (Layout). Die Alteration (*B*, *Kreuz* oder *Auflösungszeichen*) einer Note gehört zur Tonhöhe dazu und ist deshalb musikalischer Inhalt. Ob ein Versetzungszeichen (also ein *gedrucktes* Kreuz, b oder Auflösungszeichen) auch vor der Note erscheint, hängt vom Kontext, also vom Layout ab. Das Layout gehorcht bestimmten Regeln, und Versetzungszeichen werden automatisch nach diesen Regeln gesetzt. Die Versetzungszeichen im fertigen Notenbild sind nach den Regeln des Notensatzes gesetzt. Deshalb wird automatisch entschieden, wo sie erscheinen, und man muss den Ton eingeben, den man *hören* will.

In diesem Beispiel

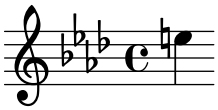
```
\key d \major
d cis fis
```



hat keine der Noten ein Versetzungszeichen, trotzdem muss im Quelltext das ‚is‘ für `cis` und `fis` notiert werden.

Der Code ‚e‘ heißt also nicht: „Zeichne einen schwarzen Punkt auf die erste Linie des Systems.“ Im Gegenteil, er heißt vielmehr: „Hier soll eine Note mit der Tonhöhe E gesetzt werden.“ In der Tonart As-Dur *bekommt* sie ein Versetzungszeichen:

```
\key aes \major
e
```



Alle diese Versetzungszeichen ausdrücklich zu schreiben, bedeutet vielleicht etwas mehr Schreibarbeit, hat aber den großen Vorteil, dass *Transpositionen* sehr viel einfacher gemacht wird und der Druck von Versetzungszeichen nach unterschiedlichen Regeln erfolgen kann. Siehe Abschnitt “Automatische Versetzungszeichen” in *Benutzerhandbuch* für einige Beispiele, wie Vorzeichen anhand von unterschiedlichen Regeln ausgegeben werden können.

Siehe auch

Benutzerhandbuch: Abschnitt “Notenbezeichnungen in anderen Sprachen” in *Benutzerhandbuch*, Abschnitt “Versetzungszeichen” in *Benutzerhandbuch*, Abschnitt “Automatische Versetzungszeichen” in *Benutzerhandbuch*, Abschnitt “Tonartbezeichnung” in *Benutzerhandbuch*.

Glossar: Abschnitt “Tonhöhenbezeichnungen” in *Glossar*.

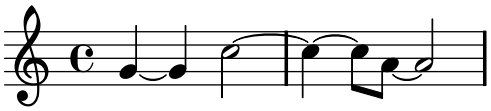
2.2.2 Bindebögen und Legatobögen

Bindebögen

Glossar: Abschnitt “Bindebogen” in *Glossar*.

Ein *Bindebogen* wird geschrieben, indem man eine Tilde ~ an die erste der zu verbindenden Noten hängt.

```
g4~ g c2~
c4 ~ c8 a8 ~ a2
```

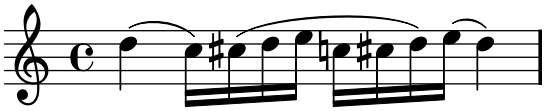


Legatobögen

Glossar: Abschnitt “Legatobogen” in *Glossar*.

Ein *Legatobogen* ist ein Bogen, der sich über mehrere Noten erstreckt. Seine Beginn- und Endnote werden mit ‚(‘ beziehungsweise ‚)‘ markiert.

```
d4( c16) cis( d e c cis d) e( d4)
```

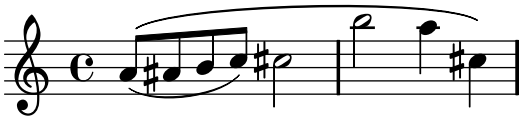


Phrasierungsbögen

Glossar: Abschnitt “Legatobogen” in *Glossar*, Abschnitt “Phrasierung” in *Glossar*.

Bögen, die längere Phrasierungseinheiten markieren (Phrasierungsbögen), werden mit `\(` und `\)` eingegeben. Es können sowohl Legato- als auch Phrasierungsbögen gleichzeitig vorkommen, aber es kann nicht mehr als jeweils einen Legato- und einen Phrasierungsbogen gleichzeitig geben.

```
a8(\( ais b c) cis2 b'2 a4 cis,\)
```

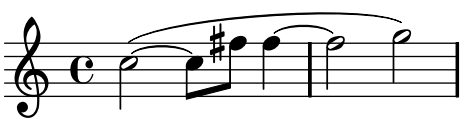


Warnung: Bindebögen sind nicht Legatobögen

Glossar: Abschnitt “Artikulationszeichen” in *Glossar*, Abschnitt “Legatobogen” in *Glossar*, Abschnitt “Bindebogen” in *Glossar*.

Ein Legatobogen sieht aus wie ein Abschnitt “Bindebogen” in *Glossar*, hat aber eine andere Bedeutung. Ein Bindebogen verlängert nur die vorhergehende Note und kann also nur bei zwei Noten gleicher Tonhöhe benutzt werden. Legatobögen dagegen zeigen die Artikulation von Noten an und können für größere Notengruppen gesetzt werden. Binde- und Legatobögen können geschachtelt werden.

```
c2~( c8 fis fis4 ~ fis2 g2)
```



Siehe auch

Benutzerhandbuch: Abschnitt “Bindebögen” in *Benutzerhandbuch*, Abschnitt “Legatobögen” in *Benutzerhandbuch*, Abschnitt “Phrasierungsbögen” in *Benutzerhandbuch*.

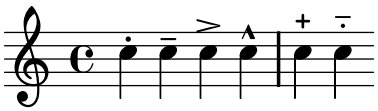
2.2.3 Artikulationszeichen und Lautstärke

Artikulationszeichen

Glossar: Abschnitt “Artikulationszeichen” in *Glossar*.

Übliche *Artikulationszeichen* können durch Anfügen von Minus (‚-‘) und einem entsprechenden Zeichen eingegeben werden:

```
c-. c-- c-> c-^ c-+ c-_
```

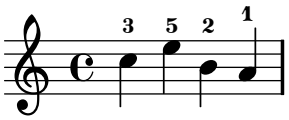


Fingersatz

Glossar: Abschnitt “Fingersatz” in *Glossar*.

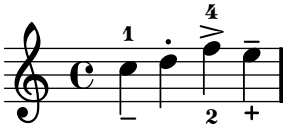
Auf gleiche Weise können Fingersatzbezeichnungen hinzugefügt werden, indem nach dem Minus (‚-‘) eine Zahl geschrieben wird:

```
c-3 e-5 b-2 a-1
```



Artikulationszeichen und Fingersätze werden normalerweise automatisch platziert, aber man kann ihre Position auch vorgeben durch die Zeichen ‚^‘ (oben) oder ‚_‘ (unten) anstelle des Minuszeichen. An eine Note können auch mehrfache Artikulationszeichen gehängt werden. Meistens findet aber LilyPond alleine die beste Möglichkeit, wie die Artikulationen platziert werden sollen.

```
c_-^1 d^. f^4_2-> e^-_+
```

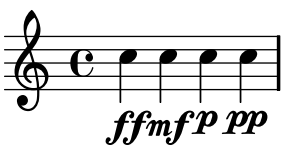


Dynamik

Glossar: Abschnitt “Dynamik” in *Glossar*, Abschnitt “Crescendo” in *Glossar*, Abschnitt “Decrescendo” in *Glossar*.

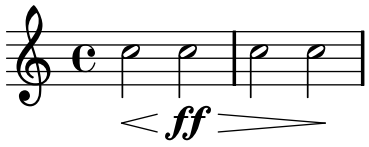
Die Dynamik innerhalb eines Stückes wird eingegeben, indem man die Markierungen (mit einem Backslash, ‚[No value for “backslash”]‘) an die Note hängt:

```
c\ff c\mf c\p c\pp
```



Crescendi und *Decrescendi* werden mit dem Befehl `\<` beziehungsweise `\>` begonnen. Das nächste absolute Dynamik-Zeichen, etwa `\f`, beendet das (De)Crescendo. Auch mit dem Befehl `\!` kann es explizit beendet werden.

```
c2\< c2\ff\> c2 c2\!
```



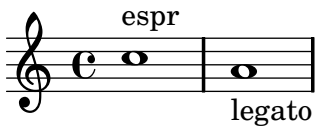
Siehe auch

Benutzerhandbuch: Abschnitt “Artikulationszeichen und Verzierungen” in *Benutzerhandbuch*, Abschnitt “Fingersatzanweisungen” in *Benutzerhandbuch*, Abschnitt “Dynamik” in *Benutzerhandbuch*.

2.2.4 Text hinzufügen

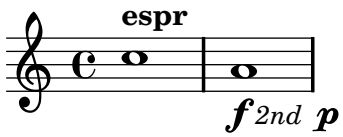
Text können Sie auf folgende Art in die Partitur einfügen:

```
c1^"espr" a_"legato"
```



Zusätzliche Formatierung kann eingesetzt werden, wenn Sie den `\markup`-Befehl benutzen:

```
c1^\markup{ \bold espr}
a1_\markup{
  \dynamic f \italic \small { 2nd } \hspace #0.1 \dynamic p
}
```



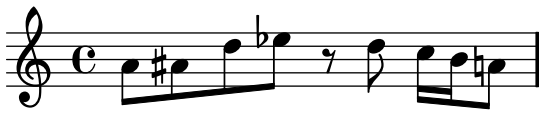
Siehe auch

Benutzerhandbuch: Abschnitt “Text eingeben” in *Benutzerhandbuch*.

2.2.5 Automatische und manuelle Balken

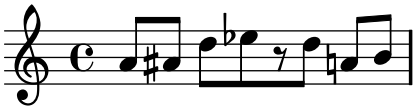
Alle *Balken* werden automatisch gesetzt:

```
a8 ais d ees r d c16 b a8
```



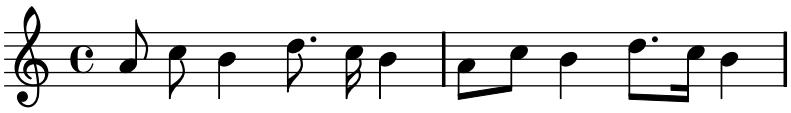
Wenn diese automatisch gesetzten Balken nicht gewollt sind, können sie manuell geändert werden. Wenn nur ein Balken hier und da korrigiert werde muss, erhält die Note, an der der Balken anfängt, ein ‚[‘ (AltGr+8) und die, an der er enden soll, ein ‚]‘ (AltGr+9).

```
a8[ ais] d[ ees r d] a b
```



Wenn Sie die automatischen Balken vollständig oder für einen längeren Abschnitt ausschalten wollen, benutzen Sie den Befehl `\autoBeamOff`, um die Balken abzuschalten, und `\autoBeamOn`, um sie wieder einzuschalten.

```
\autoBeamOff
a8 c b4 d8. c16 b4
\autoBeamOn
a8 c b4 d8. c16 b4
```



Siehe auch

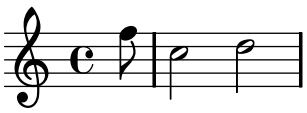
Benutzerhandbuch: Abschnitt "Automatische Balken" in *Benutzerhandbuch*, Abschnitt "Manuelle Balken" in *Benutzerhandbuch*.

2.2.6 Zusätzliche rhythmische Befehle

Auftakt

Ein *Auftakt* wird mit dem Befehl `\partial` eingegeben. Darauf folgt die Länge des Auftaktes: `\partial 4` heißt eine Viertelnote Auftakt und `\partial 8` eine Achtelnote.

```
\partial 8
f8 c2 d
```

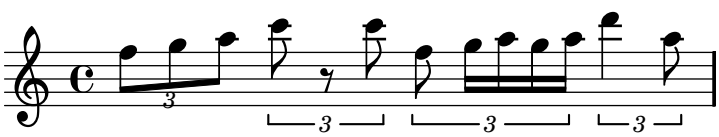


Andere rhythmische Aufteilungen

Glossar: Abschnitt "Notenwert" in *Glossar*, Abschnitt "Triole" in *Glossar*.

Triolen und *N-tolen* werden mit dem `\times`-Befehl erzeugt. Er braucht zwei Argumente: einen Bruch und die Noten, auf die er sich bezieht. Die Dauer des Abschnittes wird mit dem Bruch multipliziert. In einer Triole dauern die Noten 2/3 ihrer normalen Länge, also hat eine Triole 2/3 als Bruch:

```
\times 2/3 { f8 g a }
\times 2/3 { c r c }
\times 2/3 { f,8 g16[ a g a] }
\times 2/3 { d4 a8 }
```

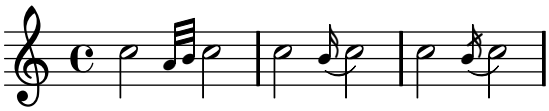


Verzierungen

Glossar: Abschnitt “Verzierungen” in *Glossar*, Abschnitt “Vorschlag” in *Glossar*, Abschnitt “Vorhalt” in *Glossar*.

Verzierungen werden mit dem Befehl `\grace` eingegeben, Vorhalte durch den Befehl `\appoggiatura` und Vorschläge mit `\acciaccatura`.

```
c2 \grace { a32[ b] } c2
c2 \appoggiatura b16 c2
c2 \acciaccatura b16 c2
```



Siehe auch

Benutzerhandbuch: Abschnitt “Verzierungen” in *Benutzerhandbuch*, Abschnitt “Andere rhythmische Aufteilungen” in *Benutzerhandbuch*, Abschnitt “Auftakte” in *Benutzerhandbuch*.

2.3 Mehrere Noten auf einmal

In diesem Kapitel wird gezeigt, wie mehr als eine Note zur gleichen Zeit gesetzt werden kann: auf unterschiedlichen Systemen für verschiedene Instrumente oder für ein Instrument (z. B. Klavier) und in Akkorden.

Polyphonie nennt man in der Musik das Vorkommen von mehr als einer Stimme in einem Stück. Polyphonie bzw. Mehrstimmigkeit heißt für LilyPond allerdings das Vorkommen von mehr als einer Stimme pro System.

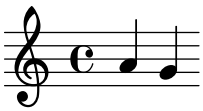
2.3.1 Musikalische Ausdrücke erklärt

In LilyPond-Quelldateien wird Musik durch *musikalische Ausdrücke* dargestellt. Eine einzelne Note ist ein musikalischer Ausdruck.

```
a4
```

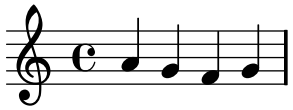
Eine Gruppe von Noten innerhalb von Klammern bildet einen neuen Ausdruck. Dieser ist nun ein *zusammengesetzter musikalischer Ausdruck*. Hier wurde solch ein zusammengesetzter musikalischer Ausdruck mit zwei Noten erstellt:

```
{ a4 g4 }
```



Wenn eine Gruppe von musikalischen Ausdrücken (also beispielsweise Noten) in geschweifte Klammern gesetzt wird, bedeutet das, dass eine Gruppe nach der anderen gesetzt wird. Das Resultat ist ein neuer musikalischer Ausdruck.

```
{ { a4 g } f g }
```



Analogie: mathematische Ausdrücke

Die Anordnung von Ausrücken funktioniert ähnlich wie mathematische Gleichungen. Eine längere Gleichung entsteht durch die Kombination kleinerer Gleichungen. Solche Gleichungen werden auch Ausdruck genannt und ihre Definition ist rekursiv, sodass beliebig komplexe und lange Ausdrücke erstellt werden können. So etwa hier:

```
1

1 + 2

(1 + 2) * 3

((1 + 2) * 3) / (4 * 5)
```

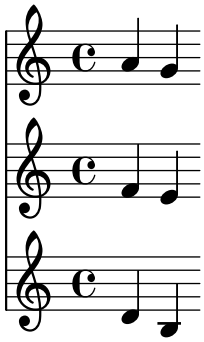
Das ist eine Folge von (mathematischen) Ausdrücken, in denen jeder Ausdruck in dem folgenden (größeren) enthalten ist. Die einfachsten Ausdrücke sind Zahlen, und größere werden durch die Kombination von Ausdrücken mit Hilfe von Operatoren (wie ‚+‘, ‚*‘ und ‚/‘) sowie Klammern. Genauso wie mathematische Ausdrücke können auch musikalische Ausdrücke beliebig tief verschachtelt werden. Das wird benötigt für komplexe Musik mit vielen Stimmen.

Gleichzeitige musikalische Ausdrücke: mehrere Notensysteme

Glossar: Abschnitt “Polyphonie” in *Glossar*.

Mit dieser Technik kann *polyphone* Musik gesetzt werden. Musikalische Ausdrücke werden einfach parallel kombiniert, damit sie gleichzeitig als eigene Stimmen in dem gleichen Notensystem gesetzt werden. Um anzuzeigen, dass an dieser Stelle gleichzeitige Noten gesetzt werden, muss nur ein Kombinationszeichen eingefügt werden. Parallel werden musikalische Ausdrücke kombiniert, indem man sie mit `<<` und `>>` einrahmt. Im folgenden Beispiel sind drei Ausdrücke (jeder mit zwei Noten) parallel kombiniert:

```
\relative c'' {
  <<
    { a4 g }
    { f e }
    { d b }
  >>
}
```



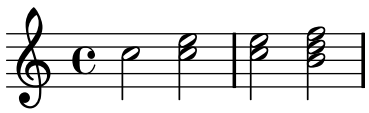
Es ist noch zu bemerken, dass wir hier für jede Ebene innerhalb der Quelldatei eine andere Einrückung geschrieben haben. Für LilyPond spielt es keine Rolle, wie viele Leerzeichen am Anfang einer Zeile sind, aber für Menschen ist es eine große Hilfe, sofort zu sehen, welche Teile des Quelltextes zusammen gehören.

> **Achtung:** Jede Note ist relativ zu der vorhergehenden in der Datei, nicht relativ zu dem zweigestrichenen C (`c''`), das im `\relative`-Befehl angegeben ist. Die Klammern haben darauf keinen Einfluss.

Gleichzeitige musikalische Ausdrücke: ein Notensystem

Um die Anzahl der Notensysteme zu bestimmen, analysiert LilyPond den Anfang des ersten Ausdrucks. Wenn sich hier eine einzelne Note befindet, wird nur ein System gesetzt, wenn es sich um eine parallele Anordnung von Ausdrücken handelt, wird mehr als ein System gesetzt. Das folgende Beispiel beginnt mit einer Note:

```
\relative c'' {
  c2 <<c e>>
  << { e f } { c <<b d>> } >>
}
```

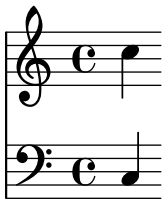


2.3.2 Mehrere Notensysteme

Wie wir in Abschnitt 2.3.1 [Musikalische Ausdrücke erklärt], Seite 26 gesehen haben, sind LilyPond-Quelldateien aus musikalischen Ausdrücken konstruiert. Wenn die Noteneingabe mit parallelen Ausdrücken beginnt, werden mehrere Notensysteme erstellt. Es ist aber sicherer und einfacherer zu verstehen, wenn diese Systeme explizit erstellt werden.

Um mehr als ein System zu schreiben, wird jedem Notenausdruck, der in einem eigenen System stehen soll, der Befehl `\new Staff` vorne angefügt. Diese `Staff` (engl. für Notensystem)-Elemente werden dann parallel angeordnet mit den `<<` und `>>`-Zeichen:

```
\relative c'' {
  <<
    \new Staff { \clef treble c }
    \new Staff { \clef bass c,, }
  >>
}
```



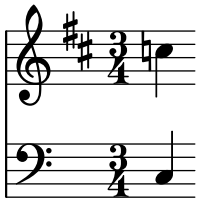
Der Befehl `\new` beginnt einen neuen „Notationskontext“. Ein solcher Notationskontext ist eine Umgebung, in der musikalische Ereignisse (wie Noten oder `\clef` (Schlüssel)-Befehle) interpretiert werden. Für einfache Stücke werden diese Umgebungen automatisch erstellt. Für kompliziertere Musik ist es aber am besten, die Umgebungen explizit zu erstellen.

Es gibt verschiedene Kontext-Typen. `Score` (Partitur), `Staff` (Notensystem) und `Voice` (Stimme) verarbeiten die Eingabe von Noten, während die `Lyrics` (Text)-Umgebung zum Setzen von Liedtexten und die `ChordNames` (Akkordbezeichnungs)-Umgebung für Akkordsymbole verwendet wird.

Indem `\new` vor einen musikalischen Ausdruck gesetzt wird, wird ein größerer Ausdruck erstellt. In diesem Sinne erinnert die Syntax des `\new`-Befehls an das Minuszeichen in der Mathematik. Genauso wie $(4 + 5)$ ein Ausdruck ist, der durch $-(4 + 5)$ zu einem größeren Ausdruck erweitert wurde, werden auch musikalische Ausdrücke durch den `\new`-Befehl erweitert.

Die Taktangabe, die in einem einzelnen System angegeben wird, wirkt sich auf alle anderen System aus. Die Angabe der Tonart in einem System hingegen beeinflusst *nicht* die Tonart der anderen Systeme. Dieses Verhalten ist darin begründet, dass Partituren mit transponierenden Instrumenten häufiger sind als Partituren mit unterschiedlichen Taktarten.

```
\relative c'' {
  <<
    \new Staff { \clef treble \key d \major \time 3/4 c }
    \new Staff { \clef bass c,, }
  >>
}
```



2.3.3 Notensysteme gruppieren

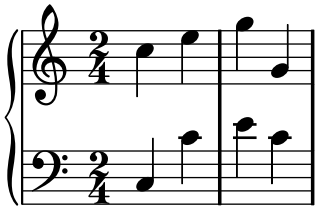
Glossar: Abschnitt “Klammer” in *Glossar*.

Musik für das Klavier wird üblicherweise auf zwei Systemen notiert, die durch eine *geschweifte Klammer* verbunden sind (Akkolade). Um ein derartiges Notensystem zu erstellen, geht man ähnlich vor wie in dem Beispiel aus Abschnitt 2.3.2 [Mehrere Notensysteme], Seite 28, nur dass der gesamte Ausdruck jetzt in eine `PianoStaff`-Umgebung eingefügt wird.

```
\new PianoStaff <<
  \new Staff ...
  \new Staff ...
>> >>
```

Hier ein kleines Beispiel:

```
\relative c'' {
  \new PianoStaff <<
    \new Staff { \time 2/4 c4 e g g, }
    \new Staff { \clef bass c,, c' e c }
  >>
}
```



Andere typische Gruppen von Notensystemen können mit den Befehlen `\new StaffGroup` für Orchestersätze und `\new ChoirStaff` für ein Chorsystem erstellt werden. Jede dieser Systemgruppen erstellt einen neuen Kontext, der dafür sorgt, dass die Klammern zu Beginn des Systems erstellt werden und der zusätzlich auch darüber entscheidet, ob die Taktlinien nur auf dem System oder auch zwischen System gesetzt werden.

Siehe auch

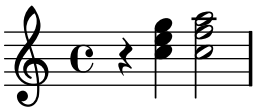
Benutzerhandbuch: Abschnitt “Tasteninstrumente” in *Benutzerhandbuch*, Abschnitt “Systeme anzeigen lassen” in *Benutzerhandbuch*.

2.3.4 Noten zu Akkorden verbinden

Glossar: Abschnitt “Akkord” in *Glossar*.

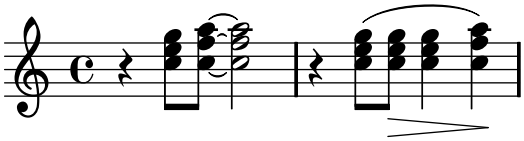
Wir haben schon weiter oben gesehen, wie Akkorde ersetellt werden können, indem sie mit spitzen Klammern eingeschlossen und somit als gleichzeitig erklingend markiert werden. Die normale Art, Akkorde zu notieren, ist aber, sie in *einfache* spitze Klammern (‚<‘ und ‚>‘) einzuschließen. Beachten Sie, dass alle Noten eines Akkordes die gleiche Dauer haben müssen, und diese Dauer wird nach der schließenden Klammer geschrieben.

```
r4 <c e g>4 <c f a>2
```



Akkorde sind im Grunde gleichwertig mit einfachen Noten: Fast alle Markierungen, die an einfache Noten angehängt werden können, kann man auch an Akkorde hängen. So können Markierungen wie Balken oder Bögen mit den Akkorden kombiniert werden. Sie müssen jedoch außerhalb der spitzen Klammern gesetzt werden.

```
r4 <c e g>8[ <c f a>]~ <c f a>2
r4 <c e g>8( <c e g>\> <c e g>4 <c f a>\!)
```

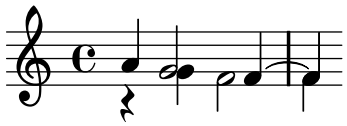


2.3.5 Mehrstimmigkeit in einem System

Wenn unterschiedliche Melodien oder Stimmen in einem System kombiniert werden sollen, werden sie als „polyphone Stimmen“ realisiert: Jede Stimme hat eigene Hälse, Balken und Legatobögen, und die Hälse der oberen Stimme zeigen immer nach oben, während die Hälse der unteren Stimme nach unten zeigen.

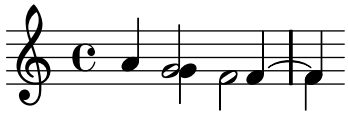
Diese Art von Notenbild wird erstellt, indem jede Stimme für sich als Abfolge notiert wird (mit `{...}`) und diese dann parallel kombiniert werden, indem die einzelnen Stimmen durch `\\` voneinander getrennt werden.

```
<<
  { a4 g2 f4~ f4 } \\
  { r4 g4 f2 f4 }
>>
```



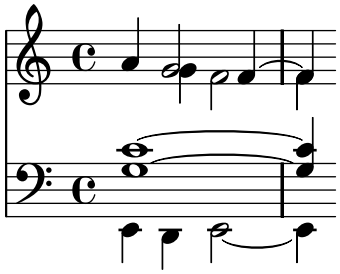
Für den Satz von mehrstimmigen Stücken kann es auch angebracht sein, unsichtbare Pausen zu verwenden. Hiermit können Stimmen ausgefüllt werden, die gerade nicht aktiv sind. Hier ist das obige Beispiel mit einer unsichtbaren Pause (‚s') anstelle einer normalen (‚r'):

```
<<
  { a4 g2 f4~ f4 } \\
  { s4 g4 f2 f4 }
>>
```



Auch diese Ausdrücke wiederum könne beliebig miteinander kombiniert werden.

```
<<
  \new Staff <<
    { a4 g2 f4~ f4 } \\
    { s4 g4 f2 f4 }
  >>
  \new Staff <<
    \clef bass
    { <c g>1 ~ <c g>4 } \\
    { e,,4 d e2 ~ e4}
  >>
>>
```



Siehe auch

Benutzerhandbuch: Abschnitt "Gleichzeitig erscheinende Noten" in *Benutzerhandbuch*.

2.4 Lieder

In diesem Kapitel wird in die Kombination von Musik mit Text eingeführt und die Erstellung einfacher Song-Blätter gezeigt.

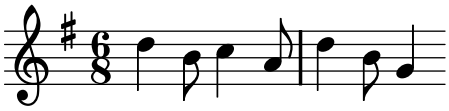
2.4.1 Einfache Lieder setzen

Glossar: Abschnitt "Gesangtext" in *Glossar*.

Hier ist der Beginn eines einfachen Kinderliedes, *Girls and boys come out to play*:

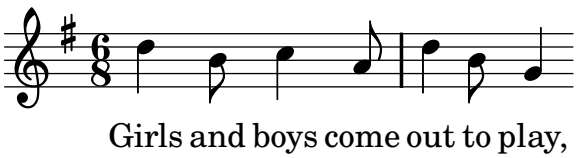
```
\relative c'' {
  \key g \major
  \time 6/8
```

```
  d4 b8 c4 a8 d4 b8 g4
}
```



Zu diesen Noten kann Text hinzugefügt werden, indem beide mit dem `\addlyrics`-Befehl kombiniert werden. Text wird eingegeben, indem jede Silbe durch ein Leerzeichen getrennt wird.

```
<<
  \relative c'' {
    \key g \major
    \time 6/8
    d4 b8 c4 a8 d4 b8 g4
  }
  \addlyrics {
    Girls and boys come out to play,
  }
>>
```



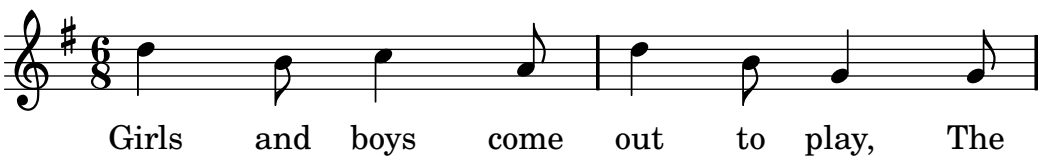
Sowohl die Noten als auch der Text sind jeweils in geschweifte Klammern eingefasst, und der gesamte Ausdruck ist zwischen `<<` ... `>>` positioniert. Damit wird garantiert, dass Text und Noten gleichzeitig gesetzt werden.

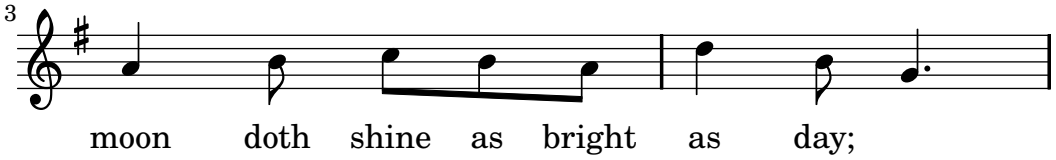
2.4.2 Text an einer Melodie ausrichten

Glossar: Abschnitt “Melisma” in *Glossar*, Abschnitt “Fülllinie” in *Glossar*.

Die nächste Zeile des Kinderliedes lautet: *The moon doth shine as bright as day*. So sieht es notiert aus:

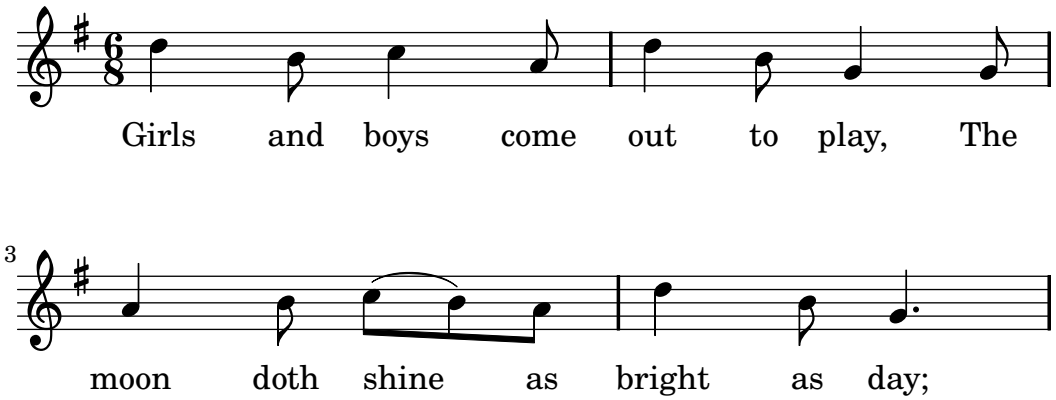
```
<<
  \relative c'' {
    \key g \major
    \time 6/8
    d4 b8 c4 a8 d4 b8 g4
    g8 a4 b8 c b a d4 b8 g4.
  }
  \addlyrics {
    Girls and boys come out to play,
    The moon doth shine as bright as day;
  }
>>
```





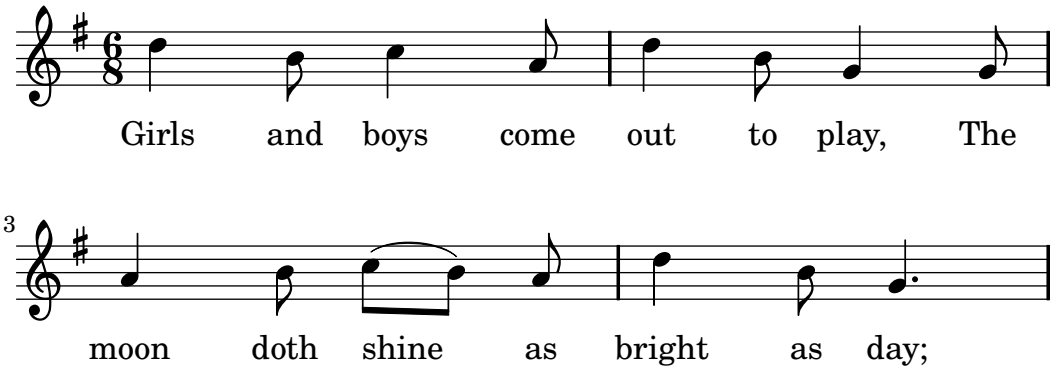
Die zusätzlichen Noten sind nicht korrekt an den Noten positioniert. Das Wort *shine* sollte eigentlich zu zwei Noten gesungen werden, nicht nur zu einer. Das wird als *Melisma* bezeichnet, wenn eine Silbe zu mehreren Noten gesungen wird. Es gibt mehrere Möglichkeiten, eine Silbe über mehrere Noten zu erweitern. Die einfachste ist, einen Legatobogen über die betroffenen Noten zu notieren, zu Einzelheiten siehe Abschnitt 2.2.2 [Bindebögen und Legatobögen], Seite 21.

```
<<
  \relative c'' {
    \key g \major
    \time 6/8
    d4 b8 c4 a8 d4 b8 g4
    g8 a4 b8 c( b) a d4 b8 g4.
  }
  \addlyrics {
    Girls and boys come out to play,
    The moon doth shine as bright as day;
  }
>>
```



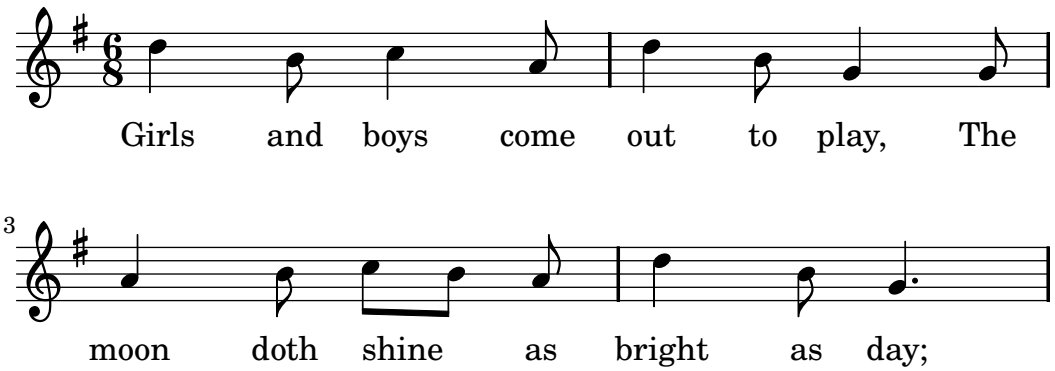
Die Wörter orientieren sich jetzt richtig an den Noten, aber der automatische Balken für die Noten zu *shine as* sieht nicht richtig aus. Wir können das korrigieren, indem wir die Balkenlänge manuell eingrenzen, damit sie der üblichen Notationsweise für Gesang entspricht. Für Einzelheiten siehe Abschnitt 2.2.5 [Automatische und manuelle Balken], Seite 24.

```
<<
  \relative c'' {
    \key g \major
    \time 6/8
    d4 b8 c4 a8 d4 b8 g4
    g8 a4 b8 c([ b]) a d4 b8 g4.
  }
  \addlyrics {
    Girls and boys come out to play,
    The moon doth shine as bright as day;
  }
>>
```



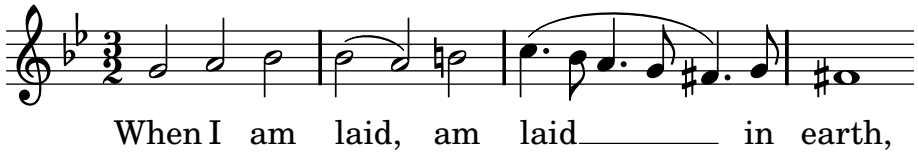
Alternativ kann das Melisma auch im Text notiert werden, indem für jede Note, die übersprungen werden soll, ein Unterstrich _ im Text geschrieben wird:

```
<<
  \relative c'' {
    \key g \major
    \time 6/8
    d4 b8 c4 a8 d4 b8 g4
    g8 a4 b8 c[ b] a d4 b8 g4.
  }
  \addlyrics {
    Girls and boys come out to play,
    The moon doth shine _ as bright as day;
  }
>>
```



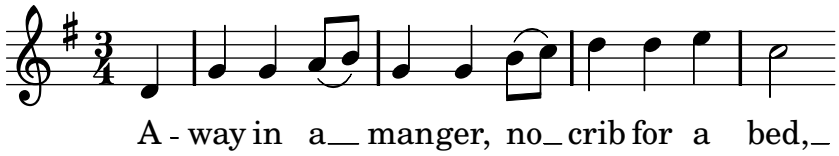
Wenn die letzte Silbe eines Wortes sich über mehrere Noten oder eine sehr lange Note erstreckt, wird üblicherweise eine Fülllinie gesetzt, die sich über alle Noten erstreckt, die zu der Silbe gehören. Diese Fülllinie wird mit zwei Unterstrichen __ notiert. Hier ein Beispiel der ersten drei Takte aus *Didos Klage*, aus Purcells *Dido and Æneas*:

```
<<
  \relative c'' {
    \key g \minor
    \time 3/2
    g2 a bes bes( a)
    b c4.( bes8 a4. g8 fis4.) g8 fis1
  }
  \addlyrics {
    When I am laid,
    am laid __ in earth,
  }
>>
```



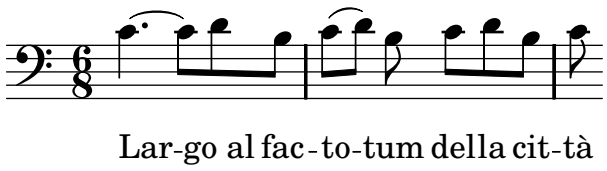
Keins der bisherigen Beispiele hat bisher Wörter benutzt, die länger als eine Silbe waren. Solche Wörter werden üblicherweise auf die Noten aufgeteilt, eine Silbe pro Note, mit Bindestrichen zwischen den Silben. Diese Silben werden durch zwei Minuszeichen notiert und von LilyPond als ein zentrierter Bindestrich zwischen den Silben gesetzt. Hier ein Beispiel, das dies und alle anderen Tricks zeigt, mit denen Text an den Noten ausgerichtet werden kann:

```
<<
  \relative c' {
    \key g \major
    \time 3/4
    \partial 4
    d4 g4 g a8( b) g4 g4
    b8( c) d4 d e4 c2
  }
  \addlyrics {
    A -- way in a __ man -- ger,
    no __ crib for a bed, __
  }
>>
```



Einige Texte, besonders in italienischer Sprache, brauchen das Gegenteil: mehr als eine Silbe muss zu einer einzelnen Note gesetzt werden. Das ist möglich, indem die Silben durch einen einzelnen Unterstrich _ zusammengekoppelt werden. Dazwischen dürfen sich keine Leerzeichen befinden, oder indem man die relevanten Silben in Anführungszeichen " setzt. Hier ein Beispiel aus dem *Figaro* von Rossini, wo die Silbe *al* auf der selben Note wie *go* des Wortes *Largo* in Figaros Arie *Largo al factotum* gesungen werden mus.

```
<<
  \relative c' {
    \clef bass
    \key c \major
    \time 6/8
    c4.~ c8 d b c([ d)] b c d b c
  }
  \addlyrics {
    Lar -- go_al fac -- to -- tum del -- la cit -- tà
  }
>>
```



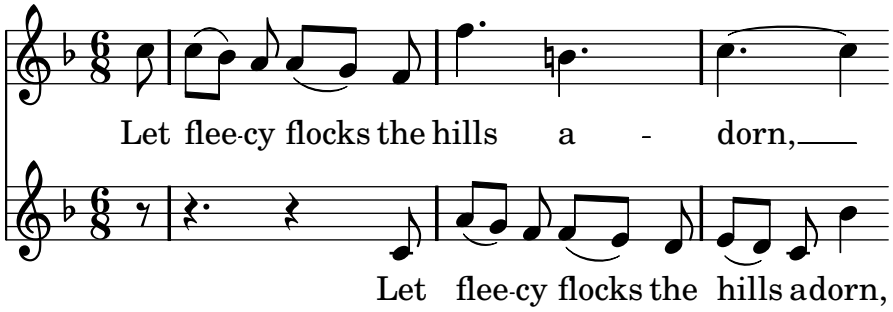
Siehe auch

Benutzerhandbuch: Abschnitt “Notation von Gesang” in *Benutzerhandbuch*.

2.4.3 Text zu mehreren Systemen

Die einfache Lösung mit `\addlyrics` kann benutzt werden, um Text zu einem oder mehreren Systemen zu setzen. Hier ein Beispiel aus Händels *Judas Maccabäus*:

```
<<
  \relative c'' {
    \key f \major
    \time 6/8
    \partial 8
    c8 c([ bes)] a a([ g)] f f'4. b, c4.~ c4
  }
  \addlyrics {
    Let flee -- cy flocks the hills a -- dorn, __
  }
  \relative c' {
    \key f \major
    \time 6/8
    \partial 8
    r8 r4. r4 c8 a'([ g)] f f([ e)] d e([ d)] c bes'4
  }
  \addlyrics {
    Let flee -- cy flocks the hills a -- dorn,
  }
>>
```



Aber Partituren, die komplizierter als dieses Beispiel sind, werden besser notiert, indem man die Systemstruktur von den Noten und dem Gesangstext durch Variablen trennt. Die Benutzung von Variablen wird erklärt im Abschnitt Abschnitt 2.5.1 [Stücke durch Bezeichner organisieren], Seite 37.

Siehe auch

Benutzerhandbuch: Abschnitt “Notation von Gesang” in *Benutzerhandbuch*.

2.5 Letzter Schliff

Das ist das letzte Kapitel der Übung. Hier soll demonstriert werden, wie man den letzten Schliff an einfachen Stücken anbringen kann. Gleichzeitig dient es als Einleitung zum Rest des Handbuches.

2.5.1 Stücke durch Bezeichner organisieren

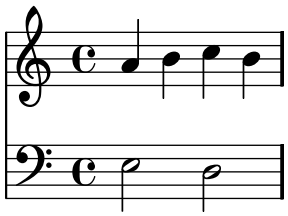
Wenn alle die Elemente, die angesprochen wurden, zu größeren Dateien zusammengefügt werden, werden auch die musikalischen Ausdrücke sehr viel größer. In polyphonen Dateien mit vielen Systemen kann das sehr chaotisch aussehen. Das Chaos kann aber deutlich reduziert werden, wenn `Variablen` definiert und verwendet werden.

Variablen (die auch als Bezeichner oder Makros bezeichnet werden) können einen Teil der Musik aufnehmen. Sie werden wie folgt definiert:

```
bezeichneteMusik = { ... }
```

Der Inhalt des musikalischen Ausdrucks `bezeichneteMusik` kann dann später wieder benutzt werden, indem man einen Backslash davor setzt (`\bezeichneteMusik`), genau wie bei jedem LilyPond-Befehl.

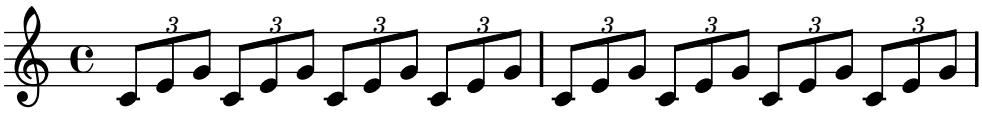
```
Geige = \new Staff
  { \relative c'' {
    a4 b c b
  }
}
Cello = \new Staff
  { \relative c {
    \clef bass
    e2 d
  }
}
{
  <<
    \Geige
    \Cello
  >>
}
```



In den Namen der Variablen dürfen nur Buchstaben des Alphabets verwendet werden, keine Zahlen oder Striche.

Variable müssen *vor* dem eigentlichen musikalischen Ausdruck definiert werden. Sie können dann aber beliebig oft verwendet werden, nachdem sie einmal definiert worden sind. Sie können sogar eingesetzt werden, um später in der Datei eine neue Variable zu erstellen. Damit kann die Schreibarbeit erleichtert werden, wenn Notengruppen sich oft wiederholen.

```
trioleA = \times 2/3 { c,8 e g }
TaktA = { \trioleA \trioleA \trioleA \trioleA }

\relative c'' {
 \TaktA \TaktA
}
```



Man kann diese Variablen auch für viele andere Objekte verwenden, etwa:

```
Breite = 4.5\cm
Name = "Tim"
aFünfPapier = \paper { paperheight = 21.0 \cm }
```

Abhängig vom Kontext kann solch ein Bezeichner in verschiedenen Stellen verwendet werden. Das folgende Beispiel zeigt die Benutzung der eben definierten Bezeichner:

```
\paper {
  \aFünfPapier
  line-width = \width
}
{
  c4^\Name
}
```

2.5.2 Versionsnummer

Der `\version`-Befehl zeigt an, welche LilyPond-Version für eine bestimmte Quelldatei benutzt worden ist:

```
\version "2.11.58"
```

Üblicherweise wird dieser Befehl am Anfang der Textdatei eingefügt.

Durch diese Versionsmarkierung werden zukünftige Aktualisierungen des LilyPond-Programmes einfacher gemacht. Syntax-Änderungen zwischen den Programmversionen werden von einem speziellen Programm, `convert-ly`, vorgenommen. Dieses Programm braucht `\version`, um zu entscheiden, welche Regeln angewandt werden müssen. Für Einzelheiten, siehe Abschnitt “Dateien mit convert-ly aktualisieren” in *Programmbenutzung*.

2.5.3 Titel hinzufügen

Titel, Komponist, Opusnummern und ähnliche Information werden in einer `\header`-Umgebung eingefügt. Diese Umgebung befindet sich außerhalb der musikalischen Ausdrücke, meistens wird die `\header`-Umgebung direkt nach der Abschnitt 2.5.2 [Versionsnummer], Seite 38 eingefügt.

```
\version "2.11.58"
\header {
  title = "Symphony"
  composer = "Ich"
  opus = "Op. 9"
}

{
  ... Noten ...
}
```

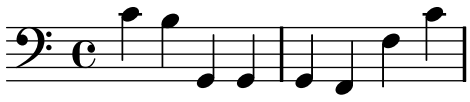
Wenn die Datei übersetzt wird, werden Titel- und Komponisteneinträge über der Musik ausgegeben. Mehr Information über die Titelei findet sich im Kapitel Abschnitt “Titel erstellen” in *Benutzerhandbuch*.

2.5.4 Absolute Notenbezeichnungen

Bis jetzt haben wir immer `\relative` benutzt, um Tonhöhen zu bestimmen. Das ist die einfachste Eingabeweise für die meiste Musik. Es gibt aber noch eine andere Möglichkeit, Tonhöhen darzustellen: durch absolute Bezeichnung.

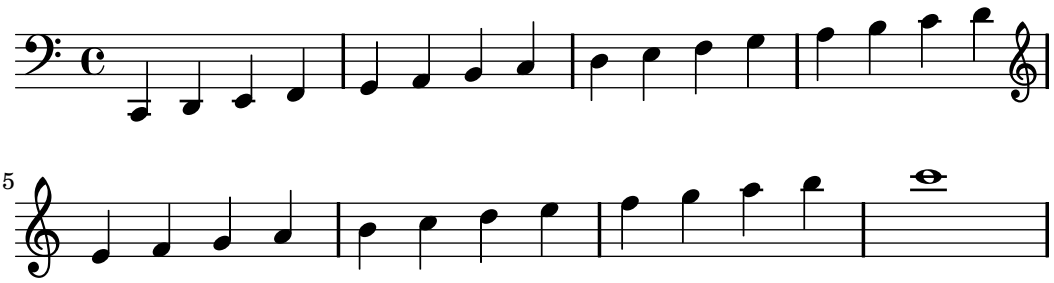
Wenn man das `\relative` weglässt, werden alle Tonhöhen von LilyPond als absolute Werte interpretiert. Ein `c'` ist dann also immer das eingestrichene C, ein `b` ist immer das kleine h unter dem eingestrichenen C, und ein `g,` ist immer das große G – also die Note auf der letzten Linie im Bass-Schlüssel.

```
{
  \clef bass
  c' b g, g,
  g, f, f c'
}
```



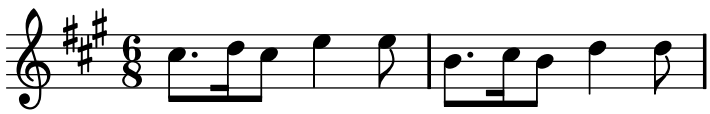
Hier eine Tonleiter über vier Oktaven:

```
{
  \clef bass
  c, d, e, f,
  g, a, b, c
  d e f g
  a b c' d'
  \clef treble
  e' f' g' a'
  b' c'' d'' e''
  f'' g'' a'' b''
  c'''1
}
```



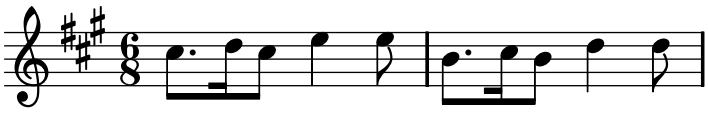
Wie leicht zu sehen ist, muss man sehr viele Apostrophe schreiben, wenn die Melodie im Sopranschlüssel notiert ist. Siehe etwa dieses Fragment von Mozart:

```
{
  \key a \major
  \time 6/8
  cis''8. d''16 cis''8 e''4 e''8
  b'8. cis''16 b'8 d''4 d''8
}
```



Alle diese Apostrophe machen den Quelltext schlecht lesbar und sind eine mögliche Fehlerquelle. Mit dem `\relative`-Befehl ist das Beispiel sehr viel einfacher zu lesen:

```
\relative c'' {
  \key a \major
  \time 6/8
  cis8. d16 cis8 e4 e8
  b8. cis16 b8 d4 d8
}
```



Wenn man einen Fehler durch ein Oktavierungszeichen (' oder ,) im \relative-Modus macht, ist er sehr schnell zu finden, denn viele Noten sind nacheinander in der falschen Oktave. Im absoluten Modus dagegen ist ein einzelner Fehler nicht so deutlich und deshalb auch nicht so einfach zu finden.

Trotz allem ist der absolute Modus gut für Musik mit sehr großen Sprüngen und vor allem für computergenerierte LilyPond-Dateien.

2.5.5 Nach der Übung

Wenn Sie diese Übung absolviert haben, sollten Sie am besten ein paar Stücke selber notieren. Beginnen Sie mit den Anhang A [Vorlagen], Seite 87 und fügen Sie einfach Ihre Noten dazu. Wenn Sie irgendetwas brauchen, das nicht in der Übung besprochen wurde, schauen Sie sich den Abschnitt Alles über die Notation an, angefangen mit Abschnitt “Musikalische Notation” in *Benutzerhandbuch*. Wenn Sie für ein Instrument oder Ensemble Noten schreiben wollen, für das es keine Vorlage gibt, schauen Sie sich Abschnitt 3.4 [Erweiterung der Beispiele], Seite 67 an.

Wenn Sie ein paar kurze Stücke notiert haben, lesen Sie den Rest des Handbuchs zum Lernen (Kapitel 3–5). Natürlich können Sie auch sofort weiterlesen. Die nächsten Kapitel sind aber mit der Annahme geschrieben, dass Sie die Eingabesprache von LilyPond beherrschen. Sie können die weiteren Kapitel auch überfliegen und dann darauf wieder zurückkommen, wenn Sie einige Erfahrungen im Notieren gewonnen haben.

In dieser Übung, genauso wie im gesamten Handbuch zum Lernen, befindet sich ein Abschnitt **Siehe auch** am Ende jedes Abschnittes, wo sich Verweise auf andere Abschnitte befinden. Diesen Verweisen sollten Sie nicht beim ersten Durchlesen folgen; erst wenn Sie das gesamte Handbuch zum Lernen gelesen haben, können Sie bei Bedarf diesen Verweisen folgen, um ein Thema zu vertiefen.

Bitte lesen Sie jetzt Abschnitt 1.2 [Über die Dokumentation], Seite 9, wenn Sie es bisher noch nicht getan haben. Es gibt ungeheuer viel Information über LilyPond, so dass Neulinge sich nicht sofort zurecht finden. Wenn Sie auch nur ein paar Minuten in diesem Abschnitt lesen, können Sie sich Stunden frustrierendes Suchen an der falschen Stelle ersparen!

3 Grundbegriffe

Nachdem im Tutorial gezeigt wurde, wie aus einfachen Text-Dateien wunderschön formatierte Musiknoten erzeugt werden können, stellt dieses Kapitel die Konzepte und Techniken vor, wie auch komplexere Partituren erstellt werden können.

3.1 Wie eine LilyPond-Datei funktioniert

Das LilyPond Eingabeformat hat eine ziemlich freie Form, so dass für erfahrene Benutzer viel Freiheit besteht, die Struktur ihrer Quelldateien anzulegen. Für Neulinge kann diese Flexibilität aber erst einmal verwirrend sein. In diesem Kapitel soll darum ein Teil dieser Strukturen dargestellt werden, vieles aber zur Vereinfachung auch weggelassen werden. Für eine komplette Beschreibung des Eingabeformats siehe Abschnitt “Die Dateistruktur” in *Benutzerhandbuch*.

Die meisten Beispiele in diesem Handbuch sind kleine Schnipsel, wie etwa dieser:

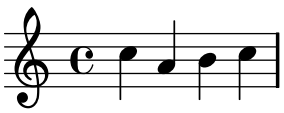
```
c4 a b c
```

Wie hoffentlich bekannt ist, lässt sich solch ein Schnipsel nicht in dieser Form übersetzen. Diese Beispiele sind also nur Kurzformen von wirklichen Beispielen. Sie müssen wenigstens zusätzlich in geschweifte Klammern gesetzt werden.

```
{
  c4 a b c
}
```

Die meisten Beispiele benutzen auch den `\relative c'`-Befehl. Der ist nicht nötig, um die Dateien zu übersetzen, aber in den meisten Fällen sieht der Notensatz seltsam aus, wenn man den Befehl weglässt.

```
\relative c'' {
  c4 a b c
}
```



Eine komplette Definition des Eingabeformats findet sich im Kapitel Abschnitt “Die Dateistruktur” in *Benutzerhandbuch*.

3.1.1 Einführung in die Dateistruktur von LilyPond

Ein grundlegendes Beispiel einer Eingabedatei für LilyPond lautet:

```
\version "2.11.58"
\header { }
\score {
  ...zusammengesetzter Musik-Ausdruck...  % Die gesamte Musik kommt hier!
  \layout { }
  \midi { }
}
```

Aufgrund der Flexibilität von LilyPond gibt es viele Variationen dieses Schemas, aber dieses Beispiel dient als einfacher Ausgangspunkt.

Bisher hat noch keines der Beispiele den `\score{}`-Befehl benutzt, da Lilypond derartige zusätzliche Befehle automatisch bei Bedarf einfügt, wenn die Eingabedatei eine einfache Struktur hat.

Sehen wir uns als ein solches einfaches Beispiel an:

```
\relative c'' {
  c4 a d c
}
```

Im Hintergrund kommen hier noch einige Ebenen dazu: LilyPond-Code in der obigen Form ist in Wirklichkeit eine Abkürzung. Auch wenn man so Dateien schreiben kann und sie auch korrekt gesetzt werden, heißt der vollständige Code, der hier gemeint ist, eigentlich:

```
\book {
  \score {
    \new Staff {
      \new Voice {
        \relative c'' {
          c4 a b c
        }
      }
    }
    \layout { }
  }
}
```

Mit anderen Worten: Wenn die Eingabedatei einen einfachen Musik-Ausdruck enthält, wird LilyPond die Datei so interpretieren, als ob dieser Ausdruck in den oben gezeigten Befehlen eingegeben wurde. Diese nötige Stuktur wird automatisch im Speicher beim Aufruf von LilyPond erzeugt, ohne dass der Benutzer davon etwas bemerkt.

Ein Wort der Warnung ist jedoch angebracht! Viele der Beispiele in der Dokumentation von LilyPond lassen die `\new Staff` und `\new Voice` Befehle zur Erzeugung einer Notenzeile und einer Stimme (beides ist in LilyPond ein sogenannter Kontext) bewusst aus, damit sie implizit von LilyPond im Speicher erzeugt werden. Für einfache Dokumente funktioniert das im Allgemeinen sehr gut, für komplexere Partituren können dadurch aber unerwartete Ergebnisse entstehen, teilweise sogar unerwartete leere Notenzeilen. Um die entsprechenden Kontexte in diesem Fall explizit zu erzeugen, siehe Abschnitt 3.3 [Kontexte und Engraver], Seite 64.

> **Achtung:** Wenn mehr als ein paar Zeilen an Musik eingegeben werden, empfielt es sich, die Notenzeilen und die Stimmen immer explizit mit `\new Staff` und `\new Voice` zu erzeugen.

Im Moment wollen wir aber zu unserem ersten Beispiel zurückkehren und nur den `\score`-Befehl näher betrachten.

Eine Partitur (`\score`) muss immer mit einem musikalischen Ausdruck beginnen. Das ist letztendlich alle Musik, angefangen bei einer einzelnen Note bis hin zu einer riesigen Partitur mit vielen Notensystemen (bezeichnet durch `GrandStaff`):

```
{
  \new GrandStaff <<
    ...hier die gesamte Partitur...
  >>
}
```

Da sich alles innerhalb der geschweiften Klammern `{ ... }` befindet, wird es wie ein einziger musikalischer Ausdruck behandelt.

Ein `\score` auch andere Dinge enthalten, wie etwa

```
\score {
  { c'4 a b c' }
  \layout { }
```

```
  \midi { }
  \header { }
}
```

Wie man sieht sind die drei Befehle `\header`, `\layout` und `\midi` von spezieller Natur: Im Gegensatz zu vielen Anderen Befehlen, die auch mit einem `\` beginnen, liefern sie *keinen* Musikausdruck und sind auch nicht Teil eines musikalischen Ausdrucks. Daher können sie sowohl innerhalb eines `\score`-Blocks als auch außerhalb plaziert werden. Tatsächlich werden einige dieser Befehle meist außerhalb des `\score`-Blocksgesetzt, zum Beispiel findet sich der `\header` sehr oft oberhalb der `\score`-Umgebung. Das funktioniert genauso gut.

Zwei bisher noch nicht aufgetauchte Befehle sind `\layout { }` und `\midi {}`. Wenn sie in einer Datei vorkommen, führt dies dazu, dass Lilypond eine druckfähige PDF-Datei bzw. eine MIDI-Datei erzeugt. Genauer beschrieben werden sie im Benutzerhandbuch – Abschnitt “Partiturlayout” in *Benutzerhandbuch* und Abschnitt “MIDI-Dateien erstellen” in *Benutzerhandbuch*.

Ihr LilyPond Code kann auch mehrere `\score`-Blöcke enthalten. Jeder davon wird als eigenständige Partitur interpretiert, die allerdings alle in dieselbe Ausgabedatei plaziert weden. Ein `\book`-Befehl ist nicht explizit notwendig – er wird implizit erzeugt. Wenn jedoch für jeden `\score`-Block in einer einzigen `.ly`-Datei eine eigene Ausgabe-Datei erzeugt werden soll, dann muss jeder dieser Blöcke in einen eigenen `\book`-Block gesetzt werden: Jeder `\book`-Block erzeugt dann eine eigene Ausgabedatei.

Zusammenfassung:

Jeder `\book`-Block erzeugt eine eigene Ausgabedatei (z.B. eine PDF-Datei). Wenn Sie keinen derartigen Block explizit angegeben haben, setzt LilyPond den gesamten Dateiinhalt innerhalb eines einzigen impliziten `\book`-Blocks.

Jeder `\score`-Block beschreibt ein eigenständiges Musikstück innerhalb des `\book`-Blocks.

Jeder `\layout`-Block wirkt sich auf den `\score`- oder `\book`-Block aus, in dem er auftritt. So wirkt z.B. ein `\layout`-Block innerhalb eines `\score`-Blocks nur auf diesen einen Block und seinen gesamten Inhalt, ein `\layout`-Block außerhalb eines `\score`-Blocks (und daher innerhalb des implizit erzeugten oder explizit angegebenen `\book`-Blocks) jedoch auf alle `\score`-Blocks innerhalb dieses `\book`-Blocks.

Nähere Details finden sich im Abschnitt Abschnitt “Mehrere Partituren in einem Buch” in *Benutzerhandbuch*.

Eine gute Möglichkeit zur Vereinfachung sind selbst definierte Variablen. Alle Vorlagen verwenden diese Möglichkeit.

```
melodie = \relative c' {
  c4 a b c
}

\score {
  { \melodie }
}
```

Wenn LilyPond diese Datei analysiert, nimmt es den Inhalt von `melodie` (alles nach dem Gleichheitszeichen) und fügt ihn immer dann ein, wenn ein `\melodie` vorkommt. Die Namen sind frei wählbar, die Variable kann genauso gut `melodie`, `GLOBAL`, `rechteHandklavier`, oder `foofoobarbaz` heißen. Für mehr Information siehe Abschnitt 5.1.4 [Tipparbeit sparen durch Bezeichner und Funktionen], Seite 78. Als Variablenname kann fast jeder beliebige Name benutzt werden, allerdings dürfen nur Buchstaben vorkommen (also keine Zahlen, Unterstriche, Sonderzeichen, etc.) und er darf nicht wie ein LilyPond-Befehl lauten. Die genauen Einschränkungen sind beschrieben in Abschnitt “Die Dateistruktur” in *Benutzerhandbuch*.

Siehe auch

Eine vollständige Definition des Eingabeformats findet sich in Abschnitt “Die Dateistruktur” in *Benutzerhandbuch*.

3.1.2 Score ist ein (einziger) zusammengesetzter musikalischer Ausdruck

Im vorigen Kapitel, Abschnitt 3.1.1 [Einführung in die Dateistruktur von LilyPond], Seite 41, wurde die allgemeine Struktur einer LilyPond-Quelldatei beschrieben. Aber anscheinend haben wir die wichtigste Frage ausgelassen, nämlich wie man herausfindet, was nach dem `\score` geschrieben werden soll.

In Wirklichkeit ist das aber gar kein Geheimnis. Diese Zeile ist die Antwort:

> *Eine Partitur fängt immer mit* `\score` *an, gefolgt von einem einzelnen musikalischen Ausdruck.*

Vielleicht wollen Sie noch einmal Abschnitt 2.3.1 [Musikalische Ausdrücke erklärt], Seite 26 überfliegen. In diesem Kapitel wurde gezeigt, wie sich große musikalische Ausdrücke aus kleinen Teilen zusammensetzen. Noten können zu Akkorden verbunden werden usw. Jetzt gehen wir aber in die andere Richtung und betrachten, wie sich ein großer musikalischer Ausdruck zerlegen lässt.

```
\score {
  {   % diese Klammer startet den großen mus. Ausdruck
    \new GrandStaff <<
      ...hier eine ganze Wagner-Oper einfügen...
    >>
  }   % diese Klammer beendet den Ausdruck
  \layout { }
}
```

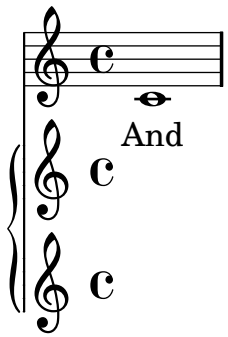
Eine Wagner-Oper ist mindestens doppelt so lang wie dieses Handbuch, beschränken wir uns also auf einen Sänger und Klavier. Wir brauchen keine Orchesterpartitur (`GrandStaff`) dafür, darum lassen wir den Befehl weg. Wir brauchen aber einen Sänger und ein Klavier.

```
\score {
  {
    <<
      \new Staff = "Sänger" <<
      >>
      \new PianoStaff = Klavier <<
      >>
    >>
  }
  \layout { }
}
```

Zur Erinnerung: mit `<<` und `>>` werden Noten gleichzeitig gesetzt; wir wollen ja auch Klavier- und Sängerstimme gleichzeitig und nicht hintereinander haben. Bei genauerem Hinsehen fällt auf, dass die `<< ... >>`-Konstruktion für die Notenzeile des Sängers eigentlich nicht unbedingt nötig wäre, da sie ja nur einen (sequenzielle) musikalischen Ausdruck enthält, nämlich alle Noten des Sängers hintereinander. Daher könnte an sich auch einfach ein `{...}` benutzt werden. Die Spitzklammern sind allerdings notwendig, sobald die Notenzeile mehrere parallelle Ausdrücke – wie etwa zwei parallele Stimmen oder eine Stimme mit zugehörigem Text – enthält. Wir werden die Musik später in das Beispiel einfügen, im Moment begnügen wir uns mit einigen Platzhalter-Noten und -Texten.

```
\score {
```

```
  <<
    \new Staff = "Sänger" <<
      \new Voice = "Singstimme" { c'1 }
      \addlyrics { And }
    >>
    \new PianoStaff = "Klavier" <<
      \new Staff = "oben" { }
      \new Staff = "unten" { }
    >>
  >>
  \layout { }
}
```



Jetzt haben wir viel mehr Details. Wir haben ein System (engl. staff) für einen Sänger, in dem sich wieder eine Stimme (engl. voice) befindet. `Voice` bedeutet für LilyPond eine Stimme (sowohl gesungen als auch gespielt) und evtl. zusätzlich einen Text. Zusätzlich werden zwei Notensysteme für das Klavier mit dem Befehl `\new PianoStaff` gesetzt. `PianoStaff` bezeichnet die Piano-Umgebung (etwa durchgehende Taktstriche und die geschweifte Klammer am Anfang), in der dann wiederum zwei eigene Systeme (`"upper"` für die rechte Hand und `"lower"` für die linke) erstellt werden.

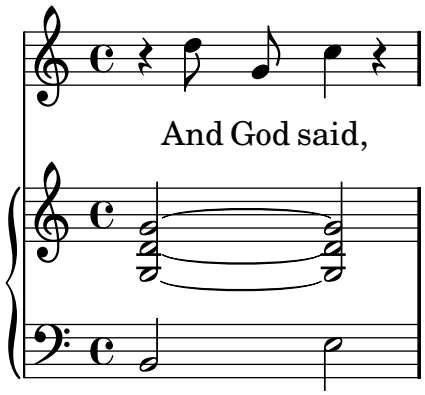
Jetzt könnte man in diese Umgebung Noten einfügen. Innerhalb der geschweiften Klammern neben `\new Voice = vocal` könnte man

```
\relative c'' {
  r4 d8\noBeam g, c4 r
}
```

schreiben. Aber wenn man seine Datei so direkt schreibt, wird der `\score`-Abschnitt sehr lang und es wird ziemlich schwer zu verstehen, wie alles zusammenhängt. Darum bietet es sich an, Bezeichner (oder Variablen) zu verwenden.

```
melodie = \relative c'' { r4 d8\noBeam g, c4 r }
Text    = \lyricmode { And God said, }
oben    = \relative c'' { <g d g,>2~ <g d g,> }
unten   = \relative c { b2 e2 }

\score {
  <<
    \new Staff = "Sänger" <<
      \new Voice = "Singstimme" { \melodie }
      \addlyrics { \Text }
    >>
    \new PianoStaff = "Klavier" <<
```

```
      \new Staff = "oben" { \oben }
      \new Staff = "unten" {
        \clef "bass"
        \unten
      }
    >>
  >>
  \layout { }
}
```



Achten Sie auf den Unterschied zwischen Noten, die mit `\relative` oder direkt in einem musikalischen Ausruck eingegeben werden, und dem Text des Lieds, der innerhalb `\lyricmode` angegeben werden muss. Diese Unterscheidung ist für LilyPond essentiell, um zu entscheiden, ob der folgende Inhalt als Musik oder Text interpretiert werden soll. Wie könnte LilyPond sonst entscheiden, ob `{a b c}` die drei Noten a, b und c darstellen soll oder den Text eines Lieds über das Alphabet!

Beim Schreiben (oder Lesen) einer `\score`-Umgebung sollte man langsam und sorgfältig vorgehen. Am besten fängt man mit dem größten Gebilde an und definiert dann die darin enthaltenen kleineren der Reihe nach. Es hilft auch, sehr genau mit den Einzügen zu sein, so dass jede Zeile, die der gleichen Ebene angehört, wirklich horizontal an der gleichen Stelle beginnt.

Siehe auch

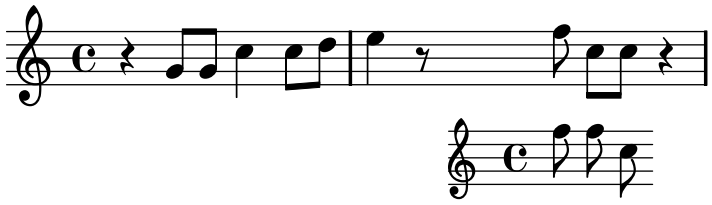
Benutzerhandbuch: Abschnitt “Struktur einer Partitur” in *Benutzerhandbuch*.

3.1.3 Musikalische Ausdrücke ineinander verschachteln

Notenzeilen (die ‚Staff‘-Kontexte) müssen nicht unbedingt gleib zu Beginn erzeugt werden – sie können auch zu einem späteren Zeitpunkt eingeführt weden. Das ist vor allem nützlichm um Abschnitt “Ossias” in *Glossar* zu erzeugen. Hier folgt ein kures Beispiel, wie eine zusätzliche temporäre Notenzeile für nur drei Noten erzeugt werden kann:

```
\new Staff {
  \relative g' {
    r4 g8 g c4 c8 d |
    e4 r8
    <<
      { f c c }
      \new Staff {
```

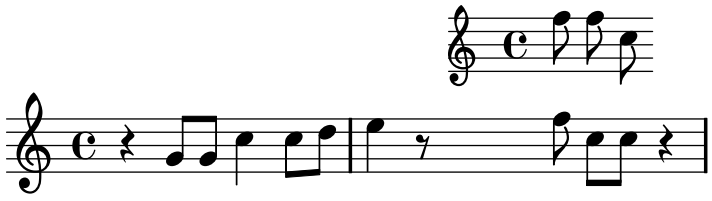
```
        f8 f c
      }
    >>
    r4 |
  }
}
```



Wie man sieht, ist die Größe des Notenschlüssels dieselbe, wie sie auch bei einer Schlüsseländerung auftritt – etwas kleiner als der Schlüssel am Beginn einer Notenzeile. Dies ist normal für Notenschlüssel, die innerhalb einer Notenzeile gesetzt werden.

Der Ossia-Abschnitt kann auch oberhalb der Hauptnotenzeile gesetzt werden:

```
\new Staff = "Hauptzeile" {
  \relative g' {
    r4 g8 g c4 c8 d |
    e4 r8
    <<
      { f c c }
      \new Staff \with {
        alignAboveContext = "Hauptzeile" }
      { f8 f c }
    >>
    r4 |
  }
}
```



Dieses Beispiel benutzt den `\with`-Befehl, der später noch genauer erklärt wird. Damit kann das Standardverhalten einer einzelnen Notenzeile geändert werden: Hier wird einfach angegeben, dass die neue Notenzeile oberhalb der bereits existierenden Zeile mit Namen „Hauptzeile“ plaziert werden soll, anstatt standardmäßig unterhalb.

Siehe auch

Ossia werden oft ohne Notenschlüssel und Taktangabe gedruck, meist auch etwas kleiner als die anderen Notenzeilen. Dies ist natürlich auch in LilyPond möglich, benötigt aber Befehle, die bisher noch nicht vorgestellt wurden. Siehe Abschnitt 4.3.2 [Größe von Objekten], Seite 68 und Abschnitt “Ossia-Systeme” in *Benutzerhandbuch*.

3.1.4 Über die Nicht-Schachtelung von Klammern und Bindebögen

Sie haben bisher zahlreiche verschiedene Arten von Klammern beim Schreiben von Musik mit LilyPond kennengelernt. Diese folgen verschiedenen Regeln, die zu Beginn vielleicht verwirrend wirken. Bevor die genauen Regeln vorgestellt werden, wollen wir die diversen Klammerarten kurz rekapitulieren:

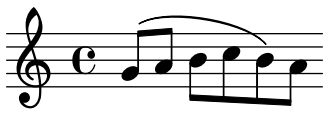
Klammerart	**Funktion**
`{ .. }`	Umschließt ein sequenzielles Musiksegment
`< .. >`	Umschließt die Noten eines Akkords
`<< .. >>`	Umschließt parallele Musikausdrücke
`( .. )`	Markiert den Beginn und das Ende eines Haltebogens
`\( .. \)`	Markiert den Beginn und das Ende eines Phasierungsbogens
`[ .. ]`	Markiert den Beginn und das Ende eines manuell erzeugten Balkens

Zusätzlich sollten vielleicht noch einige weiter Konstruktionen erwähnt werden, die Noten auf irgendeine Art und Weise verbinden: Haltebögen (durch eine Tilde `~` markiert), Triolen (als `\times x/y {..}` geschrieben) und Vorschlagnoten (als `\grace{..}` notiert).

Außerhalb von LilyPond fordert die übliche Benutzung von Klammern, dass die entsprechenden Arten korrekt verschachtelt werden, wie z.B. in `<< [ { ( .. ) } ] >>`. Die schließenden Klammern kommen dabei in der umgekehrten Reihenfolge wie die öffnenden Klammern vor. Dies ist auch in LilyPond ein **Muss** für die drei Klammerarten, die in obiger Tabelle mit dem Wort ‚Umschließt' beschrieben werden – sie müssen korrekt geschachtelt werden. Die restlichen Klammerarten (durch ‚Markiert' beschrieben), die Haltebögen und die Triolen brauchen jedoch mit den anderen Klammerarten **nicht** unbedingt korrekt geschachtelt werden. Tatsächlich sind sie auch keine Klammern in dem Sinn, dass sie etwas umschließen, sondern viel mehr Indikatoren, an welcher Stelle ein bestimmtes musikalisches Objekt beginnt oder endet.

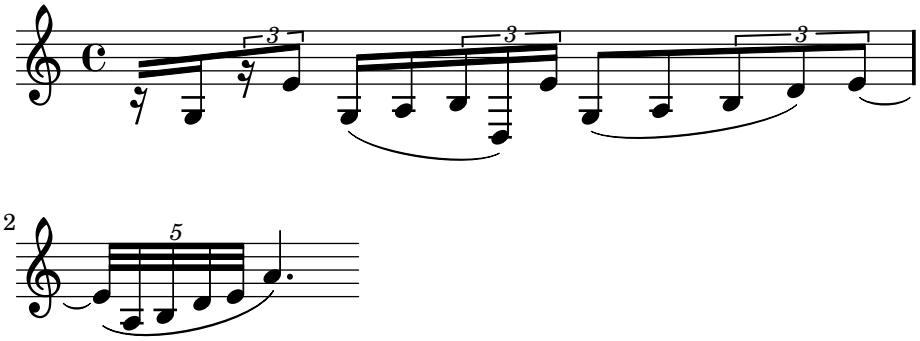
So kann also z.B. einen Phrasierungsbogen vor einem manuellen Balken beginn, jedoch schon vor dem Ende des Balkens enden. Dies mag zwar musikalisch wenig Sinn ergeben, ist aber in LilyPond auch möglich:

```
{ g8\( a b[ c b\) a] }
```



Im Allgemeinen können die verschiedenen Klammerarten, Haltebögen, Triolen und Vorschlagnoten beliebig kombiniert werden. Das folgende Beispiel zeigt einen Balken, der in eine Triole reicht (Zeile 1), eine Bindebogen, der ebenfalls in eine Triole reicht (Zeile 2), einen Balken und einen Bindeboden in eine Triole, ein Haltebogen, der über zwei Triolen läuft, sowie einen Phrasierungsbogen, der in einer Triole beginnt (Zeilen 3 und 4).

```
{
  r16[ g16 \times 2/3 {r16 e'8] }
  g16( a \times 2/3 {b d) e' }
  g8[( a \times 2/3 {b d') e'~]}
  \times 4/5 {e'32\( a b d' e'} a'4.\)
}
```



3.2 Voice enthält Noten

Sänger brauchen Stimmen zum Singen, und LilyPond braucht sie auch: in der Tat sind alle Noten für alle Instrumente in einer Partitur innerhalb von Stimmen gesetzt. Die Stimme ist das grundlegendste Prinzip von LilyPond.

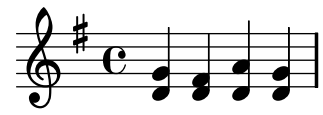
3.2.1 Ich höre Stimmen

Die grundegendsten und innersten Ebenen ein einer LilyPond-Partitur werden „Voice context“ (Stimmenkontext) oder auch nur „Voice“ (Stimme) genannt. Stimmen werden in anderen Notationsprogrammen manchmal auch als „layer“ (Ebene) bezeichnet.

Tatsächlich ist die Stimmenebene die einzige, die wirklich Noten enthalten kann. Wenn kein Stimmenkontext explizit erstellt wird, wird er automatisch erstellt, wie am Anfang dieses Kapitels gezeigt. Manche Instrumente wie etwa die Oboe können nur eine Note gleichzeitig spielen. Noten für solche Instrumente sind monophon und brauchen nur eine einzige Stimme. Instrumente, die mehrere Noten gleichzeitig spielen können, wie das Klavier, brauchen dagegeben oft mehrere Stimmen, um die verschiedenen gleichzeitig erklingenden Noten mit oft unterschiedlichen Rhythmen darstellen zu können.

Eine einzelne Stimme kann natürlich auch vielen Noten in einem Akkord enthalten – wann also braucht man dann mehrere Stimmen? Schauen wir uns zuerst dieses Beispiel mit vier Akkorden an:

```
\key g \major
<d g>4 <d fis> <d a'> <d g>
```



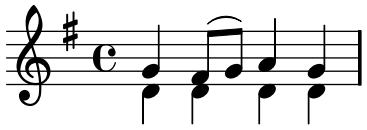
Das kann ausgedrückt werden, indem man die einfachen spitzen Klammern `< ... >` benützt, um Akkorde anzuzeigen. Hierfür braucht man nur eine Stimme. Aber gesetzt der Fall das Fis sollte eigentlich eine Achtelnote sein, gefolgt von einer Achtelnote G (als Durchgangsnote hin zum A)? Hier haben wir also zwei Noten, die zur gleichen Zeit beginnen, aber unterschiedliche Dauern haben: die Viertelnote D und die Achtelnote Fis. Wie können sie notiert werden? Als Akkord kann man sie nicht schreiben, weil alle Noten in einem Akkord die gleiche Länge besitzen müssen. Sie können auch nicht als aufeinanderfolgende Noten geschrieben werden, denn sie beginnen ja zur selben Zeit. In diesem Fall also brauchen wir zwei Stimmen.

Wie aber wird das in der LilyPond-Syntax ausgedrückt?

Die einfachste Art, Fragmente mit mehr als einer Stimme auf einem System zu notieren, ist, die Stimmen nacheinander (jeweils mit den Klammern `{ ... }`) zu schreiben und dann mit den spitzen Klammern (`<<...>>`) simultan zu kombinieren. Die beiden Fragmente müssen zusätzlich noch mit zwei Backslash-Zeichen (`\\`) voneinander getrennt werden, damit sie als zwei unterschiedliche Stimmen erkannt werden. Ohne diese Trenner würden sie als eine einzige Stimme notiert werden. Diese Technik ist besonders dann angebracht, wenn es sich bei den Noten um hauptsächlich homophone Musik handelt, in der hier und da polyphone Stellen vorkommen.

So sieht es aus, wenn die Akkorde in zwei Stimmen aufgeteilt werden und zur Durchgangsnote noch ein Bogen hinzugefügt wird:

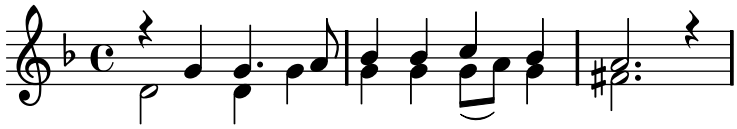
```
\key g \major
%    Voice "1"                  Voice "2"
<< { g4 fis8( g) a4 g }    \\ { d4 d d d }  >> |
```



Beachte, dass die Hälse der zweiten Stimme nun nach unten zeigen.

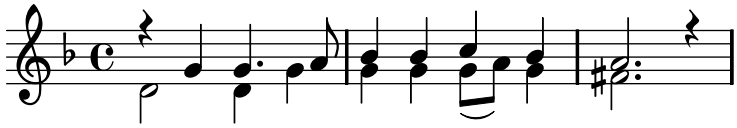
Hier ein anderes Beispiel:

```
\key d \minor
%    Voice "1"              Voice "2"
<< { r4 g g4. a8 }    \\ { d,2 d4 g }       >> |
<< { bes4 bes c bes } \\ { g4 g g8( a) g4 } >> |
<< { a2. r4 }         \\ { fis2. s4 }       >> |
```



Es ist nicht notwendig, für jeden Takt eine eigene << \\ >>-Konstruktion zu benutzen. Bei Musik mit nur wenigen Noten pro Takt kann es die Quelldatei besser lesbar machen, aber wenn in einem Takt viele Noten vorkommen, kann man die gesamten Stimmen separat schreiben, wie hier:

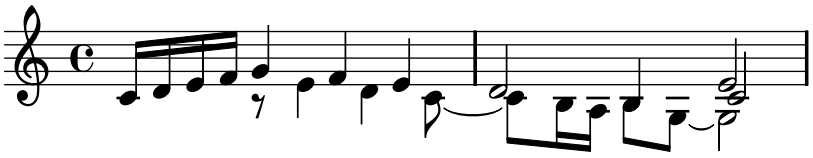
```
\key d \minor
<< {
  % Voice "1"
  r4 g g4. a8 |
  bes4 bes c bes |
  a2. r4 |
} \\ {
  % Voice "2"
  d,2 d4 g |
  g4 g g8( a) g4 |
  fis2. s4 |
} >>
```



Dieses Beispiel hat nur zwei Stimmen, aber die gleiche Konstruktion kann angewendet werden, wenn man drei oder mehr Stimmen hat, indem man weitere Backslash-Trenner hinzufügt.

Die Stimmenkontexte tragen die Namen "1", "2" usw. In jedem dieser Kontexte wird die vertikale Ausrichtung von Hälsen, Bögen, Dynamik-Zeichen usw. entsprechend ausgerichtet.

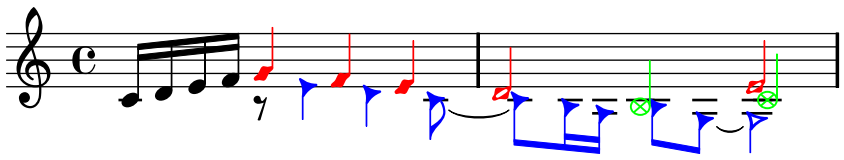
```
\new Staff \relative c' {
  % Hauptstimme
  c16 d e f
  %    Voice "1"     Voice "2"                  Voice "3"
  << { g4 f e } \\ { r8 e4 d c8 ~ } >> |
  << { d2 e2 }  \\ { c8 b16 a b8 g ~ g2 } \\ { s4 b4 c2 } >> |
}
```



Diese Stimmen sind alle getrennt von der Hauptstimme, die die Noten außerhalb der << .. >>-Konstruktion beinhaltet. Lassen wir es uns die *simultane Konstruktion* nennen. Bindebögen und Legatobögen können nur Noten in der selben Stimmen miteinander verbinden und können also somit nicht aus der simultanen Konstruktion hinausreichen. Umgekehrt gilt, dass parallele Stimmen aus eigenen simultanen Konstruktionen auf dem gleichen Notensystem die gleiche Stimme sind. Auch andere, mit dem Stimmenkontext verknüpfte Eigenschaften erstrecken sich auf alle simultanen Konstrukte. Hier das gleiche Beispiel, aber mit unterschiedlichen Farben für die Notenköpfe der unterschiedlichen Stimmen. Beachten Sie, dass Änderungen in einer Stimme sich nicht auf die anderen Stimmen erstrecken, aber sie sind weiterhin in der selben Stimme vorhanden, auch noch später im Stück. Beachten Sie auch, dass übergebundene Noten über die gleiche Stimme in zwei Konstrukten verteilt werden können, wie hier an der blauen Dreieckstimme gezeigt.

```
\new Staff \relative c' {
  % Hauptstimme
  c16 d e f
  << % Takt 1
    {
      \voiceOneStyle
      g4 f e
    }
  \\
    {
      \voiceTwoStyle
      r8 e4 d c8 ~
    }
  >>
  << % Takt 2
     % Stimme 1 geht weiter
    { d2 e2 }
  \\
     % Stimme 2 geht weiter
    { c8 b16 a b8 g ~ g2 }
  \\
    {
      \voiceThreeStyle
      s4 b4 c2
    }
  >>
```

```
}
```



Die Befehle `\voiceXXXStyle` sind vor allem dazu da, um in pädagogischen Dokumenten wie diesem hier angewandt zu werden. Sie verändern die Farbe des Notenkopfes, des Halses und des Balkens, und zusätzlich die Form des Notenkopfes, damit die einzelnen Stimmen einfach auseinander gehalten werden können. Die erste Stimme ist als rote Raute definiert, die zweite Stimme als blaue Dreiecke, die dritte Stimme als grüne Kreise mit Kreuz und die vierte Stimme (die hier nicht benutzt wird) hat dunkelrote Kreuze. `\voiceNeutralStyle` (hier auch nicht benutzt) macht diese Änderungen rückgängig. Später soll gezeigt werden, wie Befehle wie diese vom Benutzer selber erstellt werden können. Siehe auch Abschnitt 4.3.1 [Sichtbarkeit und Farbe von Objekten], Seite 68 und Abschnitt 4.7.2 [Variablen für Optimierungen einsetzen], Seite 74.

Polyphonie ändert nicht die Verhältnisse der Noten innerhalb eines `\relative { }`-Blocks. Jede Note wird weiterhin relativ zu der vorherigen Note errechnet, oder relativ zur ersten Note des vorigen Akkords. So ist etwa hier

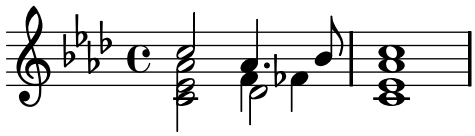
```
\relative c' { NoteA << < NoteB NoteC > \\ NoteD >> NoteE }
```

`NoteB` bezüglich `NoteA`
`NoteC` bezüglich `NoteB`, nicht `noteA`;
`NoteD` bezüglich `NoteB`, nicht `NoteA` oder `NoteC`;
`NoteE` bezüglich `NoteD`, nicht `NoteA` errechnet.

Eine andere Möglichkeit ist, den `\relative`-Befehl vor jede Stimme zu stellen. Das bietet sich an, wenn die Stimmen weit voneinander entfernt sind.

```
\relative c' { NoteA ... }
<<
  \relative c'' { < NoteB NoteC > ... }
\\
  \relative g' { NoteD ... }
>>
\relative c' { NoteE ... }
```

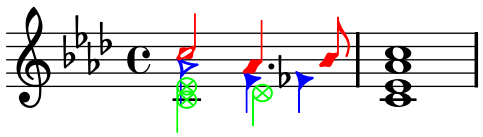
Zum Schluss wollen wir die Stimmen in einem etwas komplizierteren Stück analysieren. Hier die Noten der ersten zwei Takte von Chopins *Deux Nocturnes*, Op. 32. Dieses Beispiel soll später in diesem und dem nächsten Kapitel benutzt werden, um verschiedene Techniken, Notation zu erstellen, zu demonstrieren. Ignorieren Sie deshalb an diesem Punkt alles in folgendem Code, das Ihnen seltsam vorkommt, und konzentrieren Sie sich auf die Noten und die Stimmen. Die komplizierten Dinge werden in späteren Abschnitten erklärt werden.



Die Richtung der Hälse wird oft benutzt, um anzuzeigen, dass zwei gleichzeitige Melodien sich fortsetzen. Hier zeigen die Hälse aller oberen Noten nach oben und die Hälse aller unteren Noten nach unten. Das ist der erste Anhaltspunkt, dass mehr als eine Stimme benötigt wird.

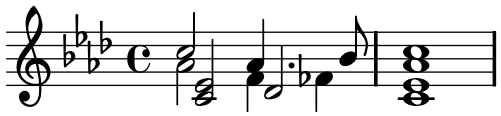
Aber die wirkliche Notwendigkeit für mehrere Stimmen tritt erst dann auf, wenn unterschiedliche Noten gleichzeitig erklingen, aber unterschiedliche Dauern besitzen. Schauen Sie

sich die Noten auf dem dritten Schlag im ersten Takt an. Das As ist eine punktierte Viertel, das F ist eine Viertel und das Des eine Halbe. Sie können nicht als Akkord geschrieben werden, denn alle Noten in einem Akkord besitzen die gleiche Dauer. Sie können aber auch nicht nacheinander geschrieben werden, denn sie beginnen auf der gleichen Taktzeit. Dieser Taktabschnitt benötigt drei Stimmen, und normalerweise schreibt man drei Stimmen für den ganzen Takt, wie im Beispiel unten zu sehen ist; hier sind unterschiedliche Köpfe und Farben für die verschiedenen Stimmen eingesetzt. Nocheinmal: der Quellcode für dieses Beispiel wird später erklärt werden, deshalb ignorieren Sie alles, was Sie hier nicht verstehen können.



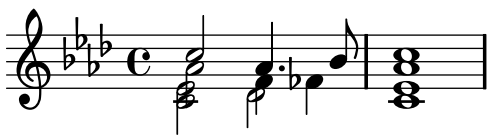
Versuchen wir also, diese Musik selber zu notieren. Wie wir sehen werden, beinhaltet das einige Schwierigkeiten. Fangen wir an, wie wir es gelernt haben, indem wir mit der << \\ >>-Konstruktion die drei Stimmen des ersten Taktes notieren:

```
\new Staff \relative c'' {
  \key aes \major
  <<
    { c2 aes4. bes8 } \\ { aes2 f4 fes } \\ { <ees c>2 des2 }
  >>
  <c ees aes c>1
}
```



Die Richtung des Notenhalses wird automatisch zugewiesen; die ungeraden Stimmen tragen Hälse nach oben, die gerade Hälse nach unten. Die Hälse für die Stimmen 1 und 2 stimmen, aber die Hälse in der dritten Stimme sollen in diesem Beispiel eigentlich nach unten zeigen. Wir können das korrigieren, indem wir die dritte Stimme einfach auslassen und die Noten in die vierte Stimme verschieben:

```
\new Staff \relative c'' {
  \key aes \major
  << % erste Stimme
    { c2 aes4. bes8 }
  \\ % zweite Stimme
    { aes2 f4 fes   }
  \\ % Stimme drei auslassen
  \\ % vierte Stimme
    { <ees c>2 des2 }
  >> |
  <c ees aes c>1 |
}
```



Wie zu sehen ist, ändert das die Richtung der Hälse, aber zeigt ein anderes Problem auf, auf das man manchmal bei mehreren Stimmen stößt: Die Hälse einer Stimme können mit den Hälsen anderer Stimmen kollidieren. LilyPond erlaubt Noten in verschiedenen Stimmen sich auf der gleichen vertikalen Position zu befinden, wenn die Hälse in entgegengesetzte Richtungen zeigen, und positioniert die dritte und vierte Stimme dann so, dass Zusammenstöße möglichst vermieden werden. Das funktioniert gewöhnlich recht gut, aber in diesem Beispiel sind die Noten der untersten Stimme eindeutig standardmäßig schlecht positioniert. LilyPond bietet verschiedene Möglichkeiten, die horizontale Position von Noten anzupassen. Wir sind aber noch nicht so weit, dass wir diese Funktionen anwenden könnten. Darum heben wir uns das Problem für einen späteren Abschnitt auf; siehe `force-hshift`-Eigenschaft in Abschnitt 4.5.2 [Überlappende Notation in Ordnung bringen], Seite 71.

Siehe auch

Notationsreferenz: Abschnitt “Mehrere Stimmen” in *Benutzerhandbuch*.

3.2.2 Stimmen explizit beginnen

Voice-Kontexte können auch manuell innerhalb eines << >>-Abschnittes initiiert werden. Mit den Befehlen `\voiceOne` bis hin zu `\voiceFour` kann jeder Stimme entsprechendes Verhalten von vertikaler Verschiebung und Richtung von Hälsen und anderen Objekten hinzugefügt werden. In längeren Partituren können die Stimmen damit besser auseinander gehalten werden.

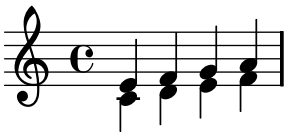
Die << \\ >>-Konstruktion, die wir im vorigen Abschnitt verwendet haben:

```
\new Staff {
  \relative c' {
    << { e4 f g a } \\ { c,4 d e f } >>
  }
}
```

ist identisch mit

```
\new Staff <<
  \new Voice = "1" { \voiceOne \relative c' { e4 f g a } }
  \new Voice = "2" { \voiceTwo \relative c' { c4 d e f } }
>>
```

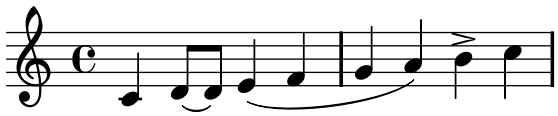
Beide würden folgendes Notenbild erzeugen:



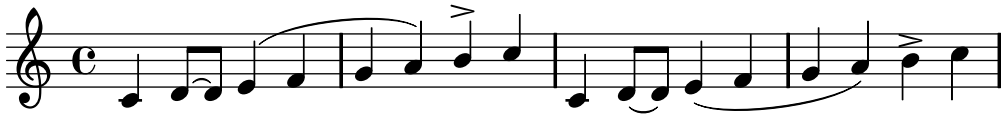
Der `\voiceXXX`-Befehl setzt die Richtung von Hälsen, Bögen, Artikulationszeichen, Text, Punktierungen und Fingersätzen. `\voiceOne` und `\voiceThree` lassen diese Objekte nach oben zeigen, `\voiceTwo` und `\voiceFour` dagegen lassen sie abwärts zeigen. Diese Befehle erzeugen eine horizontale Verschiebung, wenn es erforderlich ist, um Zusammenstöße zu vermeiden. Der Befehl `\oneVoice` stellt wieder auf das normale Verhalten um.

Schauen wir uns in einigen einfachen Beispielen an, was genau die Befehle `\oneVoice`, `\voiceOne` und `voiceTwo` mit Text, Bögen und Dynamikbezeichnung anstellen:

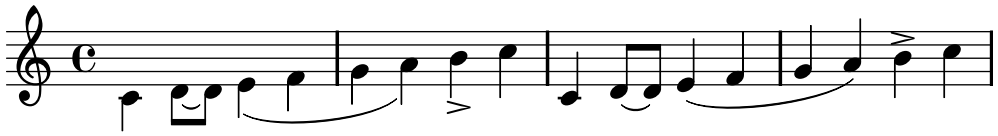
```
\relative c'{
  % Standard oder Verhalten nach \oneVoice
  c d8 ~ d e4 ( f g a ) b-> c
}
```



```
\relative c'{
  \voiceOne
  c d8 ~ d e4 ( f g a ) b-> c
  \oneVoice
  c, d8 ~ d e4 ( f g a ) b-> c
}
```



```
\relative c'{
  \voiceTwo
  c d8 ~ d e4 ( f g a ) b-> c
  \oneVoice
  c, d8 ~ d e4 ( f g a ) b-> c
}
```

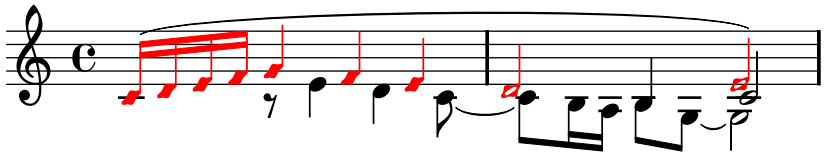


Schauen wir und nun drei unterschiedliche Arten an, den gleichen Abschnitt polyphoner Musik zu notieren, jede Art mit ihren Vorteilen in unterschiedlichen Situationen. Wir benutzen dabei das Beispiel vom vorherigen Abschnitt.

Ein Ausdruck, der direkt innerhalb einer << >>-Umgebung auftritt, gehört der Hauptstimme an. Das ist nützlich, wenn zusätzliche Stimme auftreten, während die Hauptstimme sich fortsetzt. Hier also eine bessere Version des Beispiels aus dem vorigen Abschnitt. Die farbigen Kreuz-Notenköpfe zeigen, dass die Hauptstimme sich jetzt in einem einzigen Stimmen (`voice`)-Kontext befindet. Somit kann ein Phrasierungsbogen ober sie gesetzt werden.

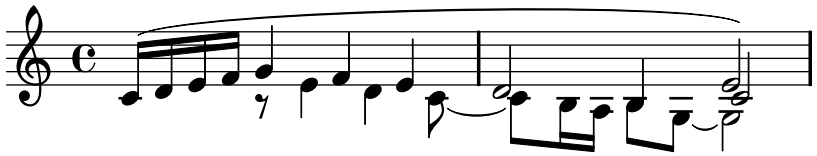
```
\new Staff \relative c' {
  \voiceOneStyle
  % Folgende Noten sind monophon
  c16^( d e f
  % Beginn von drei Stimmen gleichzeitig
  <<
    % Die Hauptstimme weiterlaufen lassen
    { g4 f e | d2 e2) }
    % Zweite Stimme einsetzen
    \new Voice {
      % Hälse usw. nach unten ausrichten
      \voiceTwo
      r8 e4 d c8 ~ | c8 b16 a b8 g ~ g2
    }
    % Die dritte Stimme beginnen
    \new Voice {
      % Hälse usw. nach oben ausrichten
      \voiceThree
      s2. | s4 b4 c2
```

```
    }
  >>
}
```



Tiefer verschachtelte polyphone Konstrukte sind möglich, und wenn eine Stimme nur kurz auftaucht, kann das der bessere Weg sein, Noten zu setzen:

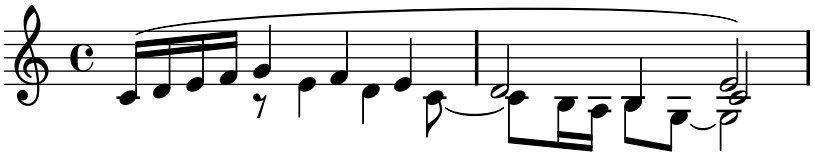
```
\new Staff \relative c' {
  c16^( d e f
  <<
    { g4 f e | d2 e2) }
    \new Voice {
      \voiceTwo
      r8 e4 d c8 ~ |
      <<
        {c8 b16 a b8 g ~ g2}
        \new Voice {
          \voiceThree
          s4 b4 c2
        }
      >>
    }
  >>
}
```



Diese Methode, neue Stimmen kurzzeitig zu verschachteln, bietet sich an, wenn nur sehr kleine Abschnitte polyphonisch gesetzt sind. Wenn aber die ganze Partitur polyphon ist, ist es meistens klarer, direkt unterschiedliche Stimmen über die gesamte Partitur hinweg einzusetzen. Hierbei kann man mit unsichtbaren Noten dann die Stellen überspringen, an denen die Stimme nicht auftaucht, wie etwa hier:

```
\new Staff \relative c' <<
  % Erste Stimme einrichten
  \new Voice {
    \voiceOne
    c16^( d e f g4 f e | d2 e2) |
  }
  % Zweite Stimme einsetzen
  \new Voice {
    % Hälse usw. nach unten ausrichten
    \voiceTwo
    s4 r8 e4 d c8 ~ | c8 b16 a b8 g ~ g2 |
  }
```

```
  % Die dritte Stimme beginnen
  \new Voice {
    % Hälse usw. nach oben ausrichten
    \voiceThree
    s1 | s4 b4 c2 |
  }
>>
```



Notenkolumnen

Dicht notierte Noten in einem Akkord, oder Noten auf der gleichen Taktzeit aber in unterschiedlichen Stimmen, werden in zwei, manchmal auch mehreren Kolumnen getzt, um die Noten am Überschneiden zu hindern. Wir bezeichnen sie als Notenkolumnen. Jede Stimme hat eine eigene Kolumne, und ein stimmenabhängiger Verschiebunsbefehl (engl. shift) wird eingesetzt, wenn eine Kollision auftreten könnte. Das zeigt das Beispiel oben. Im zweiten Takt wird das C der zweiten Stimme nach rechts verschoben, relativ gesehen zum D der ersten Stimme, und im letzten Akkord wird das C der dritten Stimme auch nach rechts verschoben im Verhältnis zu den anderen Stimmen.

Die Befehle `\shiftOn`, `\shiftOnn`, `\shiftOnnn` und `\shiftOff` bestimmen den Grad, zu dem Noten und Akkorde verschoben werden sollen, wenn sich sonst eine Kollision nicht vermeiden ließe. Die Standardeinstellung ist, dass die äußeren Stimmen (also normalerweise Stimme 1 und 2) `\shiftOff` eingestellt haben, während für die inneren Stimmen (3 und 4) `\shiftOn` eingeschaltet ist. Wenn eine Verschiebung auftritt, werden Stimmen 1 und 3 nach rechts und Stimmen 2 und 4 nach links verschoben.

`\shiftOnn` und `\shiftOnnn` definieren weitere Verschiebungsebenen, die man kurzzeitig anwählen kann, um Zusammenstöße in komplexen Situationen aufzulösen, siehe auch Abschnitt 4.5.3 [Beispiele aus dem Leben], Seite 72.

Eine Notenkolumne kann nur eine Note (oder einen Akkord) von einer Stimme mit Hälsen nach oben und eine Note (oder einen Akkord) von einer Stimme mit Hälsen nach unten tragen. Wenn Noten von zwei Stimmen mit den Hälsen in die gleiche Richtung an der selben Stelle auftreten und in beiden Stimmen ist keine Verschiebung oder die gleiche Verschiebungsebene definiert, wird die Fehlermeldung „zu viele kollidierende Notenspalten werden ignoriert“ ausgegeben.

Siehe auch

Notationsreferenz: Abschnitt “Mehrere Stimmen” in *Benutzerhandbuch*.

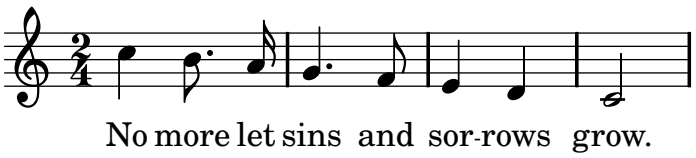
3.2.3 Stimmen und Text

Die Notation von Vokalmusik ihre eigene Schwierigkeit, nämlich die Kombination von zwei Ausdrücken: den Noten und dem Text. Achtung: Der Gesangstext wird auf Englisch „lyrics“ genannt.

Wir haben schon den `\addlyrics{}`-Befehl betrachtet, mit dem einfache Partituren gut erstellt werden können. Diese Methode ist jedoch recht eingeschränkt. Wenn der Notensatz komplexer wird, muss der Gesangstext mit einem neuen `Lyrics`-Kontext begonnen werden (mit

dem Befehl `\new Lyrics`) und durch den Befehl `\lyricsto{}` mit einer bestimmten Stimme verknüpft werden, indem die Bezeichnung der Stimme benutzt wird.

```
<<
  \new Voice = "one" \relative c'' {
    \autoBeamOff
    \time 2/4
    c4 b8. a16 g4. f8 e4 d c2
  }
  \new Lyrics \lyricsto "one" {
    No more let sins and sor -- rows grow.
  }
>>
```



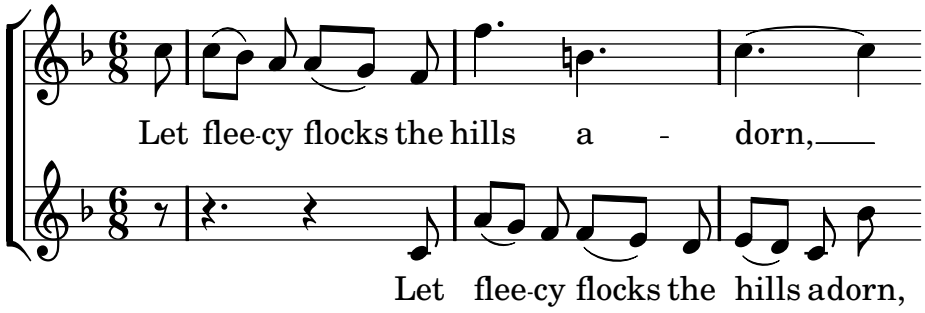
Beachten Sie, dass der Notentext nur mit einem `Voice`-Kontext verknüpft werden kann, nicht mit einem `Staff`-Kontext. In diesem Fall also müssen Sie ein System (`Staff`) und eine Stimme (`Voice`) explizit erstellen, damit alles funktioniert.

Die automatischen Balken, die LilyPond in der Standardeinstellung setzt, eignen sich sehr gut für instrumentale Musik, aber nicht so gut für Musik mit Text, wo man entweder gar keine Balken benutzt oder sie einsetzt, um Melismen zu verdeutlichen. Im Beispiel oben wird deshalb der Befehl `\autoBeamOff` eingesetzt um die automatischen Balken (engl. beam) auszuschalten.

Wir wollen das frühere Beispiel von *Judas Maccabæus* benutzen, um diese flexiblere Technik für Gesangstexte zu illustrieren. Das Beispiel wurde so umgeformt, dass jetzt Variablen eingesetzt werden, um den Text und die Noten von der Partiturstruktur zu trennen. Es wurde zusätzlich eine Chorpartiturklammer hinzugefügt. Der Gesangtext muss mit `\lyricmode` eingegeben werden, damit er als Text und nicht als Noten interpretiert werden kann.

```
global = { \time 6/8 \partial 8 \key f \major}
SoprEinsNoten = \relative c'' {
  c8 | c([ bes)] a a([ g)] f | f'4. b, | c4.~ c4 }
SoprZweiNoten = \relative c' {
  r8 | r4. r4 c8 | a'([ g)] f f([ e)] d | e([ d)] c bes' }
SopEinsText = \lyricmode {
  Let | flee -- cy flocks the | hills a -- dorn, __ }
SoprZweiText = \lyricmode {
  Let | flee -- cy flocks the | hills a -- dorn, }

\score {
  \new ChoirStaff <<
    \new Staff <<
      \new Voice = "SopOne" {
        \global
        \SoprEinsNoten
      }
      \new Lyrics \lyricsto "SopOne" {
        \SopEinsText
      }
    >>
```

```
    \new Staff <<
      \new Voice = "SopTwo" {
        \global
        \SoprZweiNoten
      }
      \new Lyrics \lyricsto "SopTwo" {
        \SoprZweiText
      }
    >>
  >>
}
```

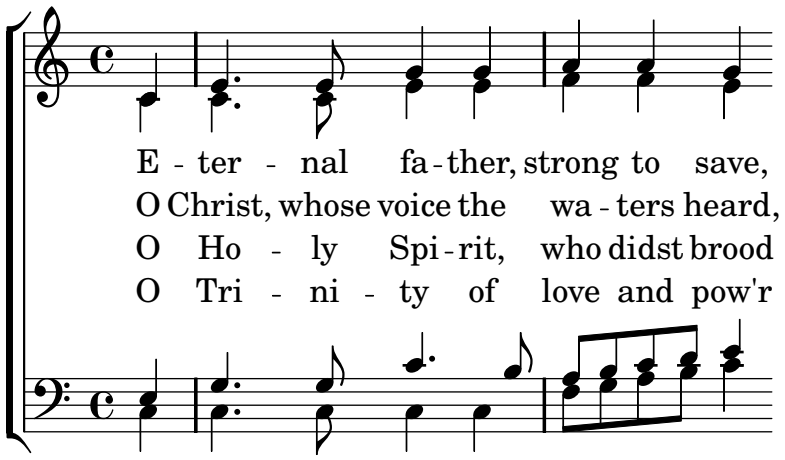


Dies ist die Grundstruktur für alle Chorpartituren. Mehr Systeme können hinzugefügt werden, wenn sie gebraucht werden, mehr Stimmen können zu jedem System hinzugefügt werden, mehr Strophen können zum Text hinzugefügt werden, und schließlich können die Variablen schnell in eine eigene Datei verschoben werden, wenn sie zu lang werden sollten.

Hier ein Beispiel der ersten Zeile eines Chorals mit vier Strophen für gemischten Chor. In diesem Fall ist der Text für alle vier Stimmen identisch. Beachten Sie, wie die Variablen eingesetzt werden, um Inhalt (Noten und Text) und Form (die Partitur) voneinander zu trennen. Eine Variable wurde eingesetzt, um die Elemente, die auf beiden Systemen auftauchen, aufzunehmen, nämlich Taktart und Tonart. Solch eine Variable wird oft auch mit „global“ bezeichnet.

```
Zeitangabe = { \time 4/4 \partial 4 \key c \major}
SoprNoten   = \relative c' { c4 | e4. e8 g4  g  | a a g }
AltNoten  = \relative c' { c4 | c4. c8 e4  e  | f f e }
TenorNoten = \relative c  { e4 | g4. g8 c4. b8 | a8 b c d e4 }
BassNoten  = \relative c  { c4 | c4. c8 c4  c  | f8 g a b c4 }
StropheEins   = \lyricmode {
  E -- | ter -- nal fa -- ther, | strong to save, }
StropheZwei   = \lyricmode {
  O | Christ, whose voice the | wa -- ters heard, }
StropheDrei = \lyricmode {
  O | Ho -- ly Spi -- rit, | who didst brood }
StropheVier  = \lyricmode {
  O | Tri -- ni -- ty of | love and pow'r }

\score {
  \new ChoirStaff <<
    \new Staff <<
      \clef "treble"
      \new Voice = "Sop"  { \voiceOne \Zeitangabe \SoprNoten }
      \new Voice = "Alto" { \voiceTwo \AltNoten }
      \new Lyrics \lyricsto "Sop" { \StropheEins   }
```

```
      \new Lyrics \lyricsto "Sop" { \StropheZwei   }
      \new Lyrics \lyricsto "Sop" { \StropheDrei }
      \new Lyrics \lyricsto "Sop" { \StropheVier  }
    >>
    \new Staff <<
      \clef "bass"
      \new Voice = "Tenor" { \voiceOne \Zeitangabe \TenorNoten }
      \new Voice = "Bass"  { \voiceTwo \BassNoten  }
    >>
  >>
}
```

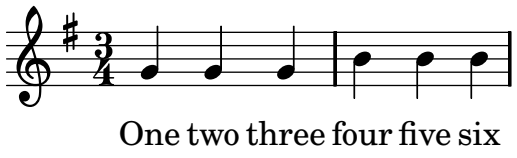


Dieser Abschnitt schließt mit einem Beispiel, das eine Solo-Strophe mit anschließendem zweistimmigem Refrain auf zwei Systemen zeigt. Die Positionierung des einstimmigen Abschnitss und der mehrstimmigen Stelle ist etwas kompliziert; es braucht etwas Aufmerksamkeit, um der Erklärung folgen zu können.

Beginnen wir mit einer `score`-Umgebung, in der eine Chorpartitur (`ChoirStaff`) gesetzt wird. Die Partitur soll schließlich mit der eckigen Klammer beginnen. Normalerweise bräuchten wir spitze Klammern im Quelltext nach dem `\new ChoirStaff`, damit die Systeme paralell gesetzt werden, aber hier wollen wir diese Parallelsierung ja erst nach dem Solo. Also benutzen wir geschweifte Klammern. Innerhalb der Chorpartitur erstellen wir zuerst das System, das die Strophe enthält. Es braucht Noten und Text parallel, also setzen wir hier die spitzen Klammern um `\new Voice` und `\new Lyrics`.

```
StrophenNoten = \relative c'' {
  \clef "treble"
  \key g \major
  \time 3/4 g g g b b b
}
SrophenText = \lyricmode {
  One two three four five six
}
\score {
  \new Choirstaff {
    \new Staff <<
      \new Voice = "verse" {
        \StrophenNoten \break
      }
      \new Lyrics \lyricsto verse {
```

```
        \SrophenText
      }
    >>
  }
}
```



Damit erhalten wir die Strophe.

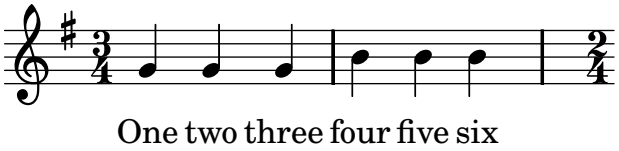
Jetzt soll *refrainA* auf dem selben System gesetzt werden, während gleichzeitig in einem neuen System darunter *refrainB* gesetzt wird. Damit die Oberstimme das gleiche System benutzt, muss alles direkt auf den Zeilenumbruchbefehl (`\break` folgen, innerhalb der *verse*-Stimme. Ja, tatsächlich, *innerhalb* der *verse*-Stimme. Hier haben wir diese parallele Stelle isoliert. Weitere Systeme könnten auf die gleiche Weise hinzugefügt werden.

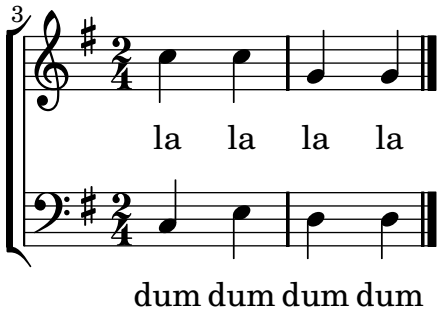
```
<<
  \refrainnotesA
  \new Lyrics \lyricsto verse {
    \refrainwordsA
  }
  \new Staff <<
    \new Voice = "refrainB" {
      \refrainnotesB
    }
    \new Lyrics \lyricsto "refrainB" {
      \refrainwordsB
    }
  >>
>>
```

Nun schließlich das Resultat mit zwei Systemen für den Refrain, man kann gut sehen, wie sich die parallele Stelle innherhalb der *verse*-Stimme befindet.

```
StrophenNoten = \relative c'' {
  \clef "treble"
  \key g \major
  \time 3/4 g g g b b b
}
RefrainNotenA = \relative c'' {
  \time 2/4
  c c g g \bar "|."
}
RefrainNotenB = \relative c {
  \clef "bass"
  \key g \major
  c e d d
}
SrophenText = \lyricmode {
  One two three four five six
}
RefrainTextA = \lyricmode {
```

```
  la la la la
}
RefrainTextB = \lyricmode {
  dum dum dum dum
}
\score {
  \new ChoirStaff {
    \new Staff <<
      \new Voice = "verse" {
        \StrophenNoten \break
        <<
          \RefrainNotenA
          \new Lyrics \lyricsto "verse" {
            \RefrainTextA
          }
          \new Staff <<
            \new Voice = "refrainB" {
              \RefrainNotenB
            }
            \new Lyrics \lyricsto "refrainB" {
              \RefrainTextB
            }
          >>
        >>
      }
      \new Lyrics \lyricsto "verse" {
        \SrophenText
      }
    >>
  }
}
```

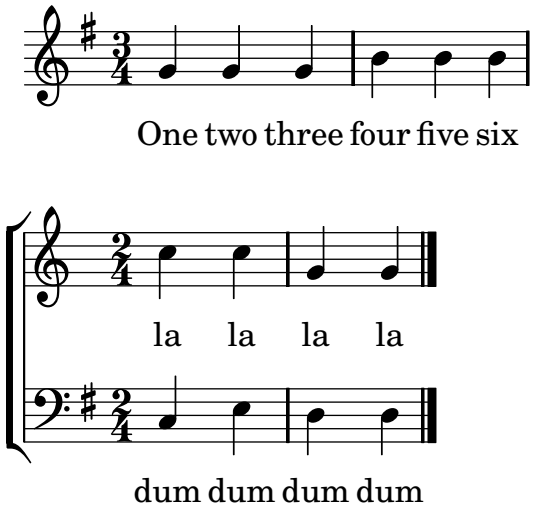




Dies ist zwar eine interessante und nützliche Übung um zu verstehen, wie sequentielle und parallele Notationsumgebungen funktionieren, in der Praxis würde man diesen Code aber vielleicht eher in zwei `\score`-Umgebungen trennnen und diese dann innerhalb einer `\book`-Umgebung einsetzen, wie im folgenden Beispiel demonstriert:

```
StrophenNoten = \relative c'' {
  \clef "treble"
  \key g \major
  \time 3/4 g g g b b b
}
RefrainNotenA = \relative c'' {
  \time 2/4
  c c g g \bar "|."
}
RefrainNotenB = \relative c {
  \clef "bass"
  \key g \major
  c e d d
}
SrophenText = \lyricmode {
  One two three four five six
}
RefrainTextA = \lyricmode {
  la la la la
}
RefrainTextB = \lyricmode {
  dum dum dum dum
}
\score {
  \new Staff <<
    \new Voice = "verse" {
      \StrophenNoten
    }
    \new Lyrics \lyricsto "verse" {
      \SrophenText
    }
  >>
}

\score {
  \new ChoirStaff <<
    \new Staff <<
      \new Voice = "refrainA" {
        \RefrainNotenA
      }
      \new Lyrics \lyricsto "refrainA" {
        \RefrainTextA
      }
    >>
    \new Staff <<
      \new Voice = "refrainB" {
        \RefrainNotenB
      }
      \new Lyrics \lyricsto "refrainB" {
        \RefrainTextB
      }
    >>
```

```
  >>
}
```



Siehe auch

Notation Reference: Abschnitt “Notation von Gesang” in *Benutzerhandbuch*.

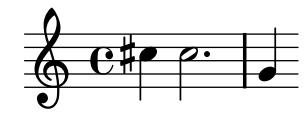
3.3 Kontexte und Engraver

Kontexte und Engraver („Stempel“) sind in den vorherigen Abschnitten schon aufgetaucht; hier wollen wir uns ihnen nun etwas ausführlicher widmen, denn sie sind sehr wichtig, um Feineinstellungen in der LilyPond-Notenausgabe vornehmen zu können.

3.3.1 Was sind Umgebungen?

Wenn Noten gesetzt werden, müssen viele Elemente zu der Notenausgabe hinzugefügt werden, die im Quellcode gar nicht explizit vorkommen. Vergleichen Sie etwa den Quellcode und die Notenausgabe des folgenden Beispiels:

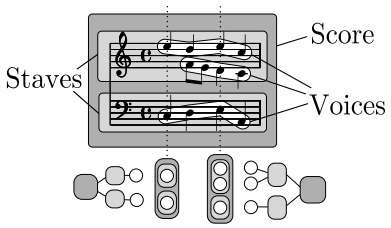
```
cis4 cis2. g4
```



Der Quellcode ist sehr kurz und knapp, während in der Notenausgabe Taktlinien, Vorzeichen, ein Schlüssel und eine Taktart hinzugefügt wurden. Während LilyPond den Eingabetext *interpretiert*, wird die musikalische Information in zeitlicher Reihenfolge inspiziert, etwa wie man eine Partitur von links nach rechts liest. Während das Programm den Code liest, merkt es sich, wo sich Taktgrenzen befinden und für welche Tonhöhen Versetzungszeichen gesetzt werden müssen. Diese Information muss auf mehreren Ebenen gehandhabt werden, denn Versetzungszeichen etwa beziehen sich nur auf ein System, Taktlinien dagegen üblicherweise auf die gesamte Partitur.

Innerhalb von LilyPond sind diese Regeln und Informationshappen in *Kontexten* (engl. contexts) gruppiert. Wir sind schon auf den `Voice` (Stimmen)-Kontext gestoßen. Daneben gibt es noch die `Staff` (Notensystem-) und `Score` (Partitur-) Kontexte. Kontexte sind hierarchisch geschichtet um die hierarchische Struktur einer Partitur zu spiegeln. Ein `Staff`-Kontext

kann zum Beispiel viele `Voice`-Kontexte beinhalten, und ein `Score`-Kontext kann viele `Staff`-Kontexte beinhalten.



Jeder Kontext hat die Aufgabe, bestimmte Notationsregeln zu erzwingen, bestimmte Notationsobjekte zu erstellen und verbundene Elemente zu ordnen. Der `Voice`-Kontext zum Beispiel kann eine Vorzeichenregel einführen und der `Staff`-Kontext hält diese Regel dann aufrecht, um einzuordenen, ob ein Versetzungszeichen gesetzt werden muss oder nicht.

Ein anderes Beispiel: die Synchronisation der Taktlinien ist standardmäßig im `Score`-Kontext verankert. Manchmal sollen die Systeme einer Partitur aber unterschiedliche Taktarten enthalten, etwa in einer polymetrischen Partitur mit 4/4- und 3/4-Takt. In diesem Fall müssen also die Standardeinstellungen der `Score`- und `Staff`-Kontexte verändert werden.

In einfachen Partituren werden die Kontexte implizit erstellt, und es kann sein, dass Sie sich dessen gar nicht bewusst sind. Für etwas größere Projekte, etwa mit vielen Systemen, müssen die Kontexte aber explizit erstellt werden, um sicher zu gehen, dass man auch wirklich die erwünschte Zahl an Systemen in der richtigen Reihenfolge erhält. Wenn Stücke mit spezialisierter Notation gesetzt werden sollen, ist es üblich, die existierenden Kontexte zu verändern oder gar gänzlich neue zu definieren.

Zusätzlich zu den `Score, Staff` und `Voice`-Kontexten gibt es noch Kontexte, die zwischen der Partitur- und Systemebene liegen und Gruppen von Systemen kontrollieren. Das sind beispielsweise der `PianoStaff` und `ChoirStaff`-Kontext. Es gibt zusätzlich alternative Kontexte für Systeme und Stimmen sowie eigene Kontexte für Gesangstexte, Perkussion, Griffsymbole, Generalbass usw.

Die Bezeichnungen all dieser Kontexte werden von einem oder mehreren englischen Wörtern gebildet, dabei wird jedes Wort mit einem Großbuchstaben begonnen und direkt an das folgende ohne Bindestrich oder Unterstrich angeschlossen, etwa `GregorianTranscriptionStaff`.

Siehe auch

Notationreferenz: Abschnitt “Was sind Umgebungen?” in *Benutzerhandbuch*.

3.3.2 Umgebungen erstellen

Es gibt nur einen Kontext der obersten Ebene: der `Score`-Kontext. Er wird mit dem `\score`-Befehl, oder – in einfacheren Partituren – automatisch erstellt.

Wenn nur ein System vorhanden ist, kann man es ruhig LilyPond überlassen, die `Voice`- und `Staff`-Kontexte zu erstellen, aber für komplexere Partituren ist es notwendig, sie mit einem Befehl zu erstellen. Der einfachste Befehl hierzu ist `\new`. Er wird dem musikalischen Ausdruck vorangestellt, etwa so:

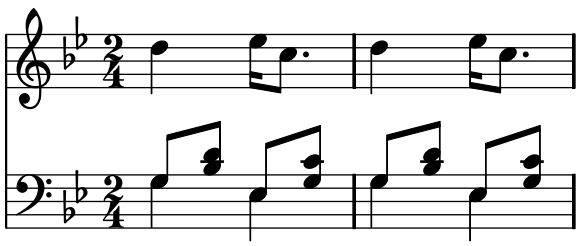
```
\new Typ musikalischer Ausdruck
```

wobei *Typ* eine Kontextbezeichnung (wie etwa `Staff` oder `Voice`) ist. Dieser Befehl erstellt einen neuen Kontext und beginnt, den *muskalischen Ausdruck* innherhalb dieses Kontexts auszuwerten.

Beachten Sie, dass es keinen `\new Score`-Befehl gibt: der Partitur-Kontext der obersten Ebene wird mit dem Befehl `\score` begonnen.

Wir haben schon viele explizite Beispiel gesehen, in denen neue `Staff`- und `Voice`-Kontexte erstellt wurden, aber um noch einmal ins Gedächtnis zu rufen, wie diese Befehle benutzt werden, hier ein kommentiertes Beispiel aus dem richtigen Leben:

```
\score {  % Beginn des einen musikalischen Ausdrucks
  <<  % Beginn von gleichzeitigen Systemen
    \time 2/4
    \new Staff {  % RH-System erstellen
      \key g \minor
      \clef "treble"
      \new Voice {  % Stimme für RH Noten erstellen
        \relative c'' {  % Beginn von RH Noten
          d4 ees16 c8. |
          d4 ees16 c8. |
        }  % Ende RH-Noten
      }  % Ende der RH Stimme
    }  % Ende RH-System
    \new Staff <<  % LH System erstellen, braucht zwei gleichzeitige Stimmen
      \key g \minor
      \clef "bass"
      \new Voice {  % LH Stimme eins erstellen
        \voiceOne
        \relative g {  % Beginn von LH Stimme eins Noten
          g8 <bes d> ees, <g c> |
          g8 <bes d> ees, <g c> |
        }  % Ende von LH Stimme eins Noten
      }  % Ende LH Stimme eins
      \new Voice {  % LH Stimme zwei erstellen
        \voiceTwo
        \relative g {  % Beginn von LH Stimme zwei Noten
          g4 ees |
          g4 ees |
        }  % Ende der LH Stimme zwei Noten
      }  % Ende der LH Stimme zwei
    >>  % Ende LH System
  >>  % Ende der gleichzeitigen Systeme
}  % Ende des einen zusammengesetzten Musikausdrucks
```



(Beachten Sie, dass wir hier alle Zeilen, die eine neue Umgebung entweder mit einer geschweiften Klammer (`{`) oder doppelten spitzen Klammern (`<<`) öffnen, mit jeweils zwei Leerzeichen, und die entsprechenden schließenden Klammern mit der gleichen Anzahl Leerzeichen eingerückt werden. Dies ist nicht erforderlich, es wird aber zu einem großen Teil die nicht passenden Klammerpaar-Fehler eliminieren und ist darum sehr empfohlen. Es macht es möglich, die Struktur einer Partitur auf einen Blick zu verstehen, und alle nicht passenden Klammern

erschließen sich schnell. Beachten Sie auch, dass das untere Notensystem mit eckigen Klammern erstellt wird, denn innerhalb dieses Systems brauchen wir zwei Stimmen, um die Noten darzustellen. Das obere System braucht nur einen einzigen musikalischen Ausdruck und ist deshalb von geschweiften Klammern umschlossen.)

Der `\new`-Befehl kann einem Kontext auch einen Namen zur Identifikation geben, um ihn von anderen Kontexten des selben Typs zu unterscheiden:

```
\new Typ = Name musikalischer Ausdruck
```

Beachten Sie den Unterschied zwischen der Bezeichnung des Kontexttyps (`Staff`, `Voice`, usw.) und dem Namen, der aus beliebigen Buchstaben (**Achtung: keine Zahlen**) bestehen kann und vom Benutzer frei erfunden werden kann. Der Name wird benutzt, um später auf genau diesen spezifischen Kontext zu verweisen. Dieses Vorgehen wurde schon in dem Abschnitt zu Gesangstexten angewandt, siehe Abschnitt 3.2.3 [Stimmen und Text], Seite 57.

Siehe auch

Notationreferenz: Abschnitt “Umgebungen erstellen” in *Benutzerhandbuch*.

3.3.3 Was sind Engraver?

3.3.4 Umgebungs-Eignschaften verändern

3.3.5 Engraver hinzufügen und entfernen

3.4 Erweiterung der Beispiele

3.4.1 Sopran und Cello

3.4.2 Vierstimmige SATB-Partitur

3.4.3 Eine Partitur von Grund auf erstellen

4 Die Ausgabe verändern

In diesem Kapitel wird erklärt, wie man die Notenausgabe verändern kann. In LilyPond kann man sehr viel konfigurieren, fast jedes Notenfragment kann geändert werden.

4.1 Grundlagen für die Optimierung

4.1.1 Grundlagen zur Optimierung

4.1.2 Objekte und Schnittstellen

4.1.3 Regeln zur Benennung von Objekten und Eigenschaften

4.1.4 Optimierungsmethoden

4.2 Die Referenz der Programminterna

4.2.1 Eigenschaften von Layoutobjekten

4.2.2 Eigenschaften, die Schnittstellen besitzen können

4.2.3 Typen von Eigenschaften

4.3 Erscheinung von Objekten

4.3.1 Sichtbarkeit und Farbe von Objekten

4.3.2 Größe von Objekten

4.3.3 Länge und Dicke von Objekten

4.4 Positionierung von Objekten

4.4.1 Automatisches Verhalten

4.4.2 withing-staff (Objekte innerhalb des Notensystems)

4.4.3 Objekte außerhalb des Notensystems

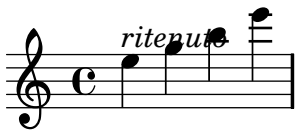
4.5 Kollision von Objekten

4.5.1 Verschieben von Objekten

Es wird vielleicht eine Überraschung sein, aber LilyPond ist nicht perfekt. Einige Notationselemente können sich überschneiden. Das ist nicht schön, kann aber (in den meisten Fällen) sehr einfach korrigiert werden.

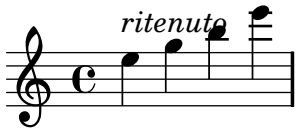
TODO: Mit den neuen Abstandseigenschaften seit Version 2.12 sind die jeweiligen Beispiele nicht mehr relevant. Sie zeigen jedoch immer noch machtvolle Eigenschaften von LilyPond und verbleiben deshalb in der Dokumentation, bis jemand bessere Beispiel zur Verfügung stellt.

```
    % temporary code to break this example:
    \override TextScript #'outside-staff-priority = ##f
  e4^\markup{ \italic ritenuto } g b e
```



Die einfachste Lösung ist es, Abstände zwischen Objekt und Note zu vergrößern (genauso auch für Fingersätze oder Dynamikzeichen). In LilyPond wird das durch Veränderung der `padding` (Füllungs)-Eigenschaft erreicht, ihre Maßeinheit sind Notenzeilenabstände. Für die meisten Objekte ist der Wert etwa 1.0 oder weniger (das unterscheidet sich von Objekt zu Objekt). Hier soll der Abstand vergrößert werden, also scheint 1.5 eine gute Wahl.

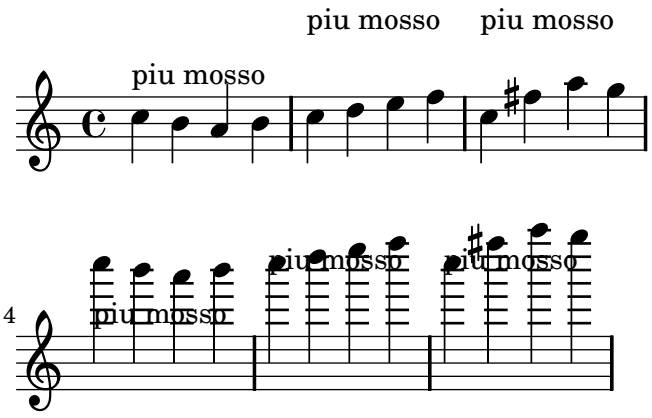
```
  % temporary code to break this example:
  \override TextScript #'outside-staff-priority = ##f
\once \override TextScript #'padding = #1.5
e4^\markup{ \italic ritenuto } g b e
```



Das sieht besser aus, ist aber noch nicht groß genug. Nach einigen Experimenten wird darum 2.3 genommen für diesen Fall. Diese Zahl ist aber nur das Resultat einigen Probierens und persönlicher Geschmack. Probieren Sie selber ein wenig herum und entscheiden Sie nach eigenem Geschmack.

Die `staff-padding`-Eigenschaft ist der vorigen sehr ähnlich. `padding` entscheidet über den minimalen Abstand zwischen einem Objekt und dem nächsten anderen Objekt (meistens eine Note oder Notenzeile); `staff-padding` entscheidet über den minimalen Abstand zwischen einem Objekt und dem Notensystem. Das ist nur ein kleiner Unterschied, aber hier wird das Verhalten demonstriert:

```
  % temporary code to break this example:
  \override TextScript #'outside-staff-priority = ##f
c4^"piu mosso" b a b
\once \override TextScript #'padding = #4.6
c4^"piu mosso" d e f
\once \override TextScript #'staff-padding = #4.6
c4^"piu mosso" fis a g
\break
c'4^"piu mosso" b a b
\once \override TextScript #'padding = #4.6
c4^"piu mosso" d e f
\once \override TextScript #'staff-padding = #4.6
c4^"piu mosso" fis a g
```

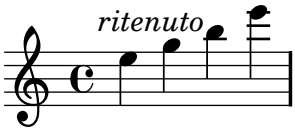


Eine andere Lösung ermöglicht vollständige Kontrolle über die Positionierung eines Objektes sowohl horizontal als auch vertikal. Das wird mit der `extra-offset` (Zusätzlicher-Abstand)-Eigenschaft erreicht. Das ist etwas komplizierter und kann andere Probleme mit sich ziehen. Wenn Objekte mit dieser Eigenschaft verschoben werden, heißt das, dass LilyPond sie erst setzt, nachdem alle anderen Objekte positioniert worden sind. Deshalb können sich die Objekte am Ende überlagern.

```
  % temporary code to break this example:
  \override TextScript #'outside-staff-priority = ##f
\once \override TextScript #'extra-offset = #'( 1.0 . -1.0 )
e4^\markup{ \italic ritenuto } g b e
```

Bei Verwendung von `extra-offset` bestimmt die erste Zahl über die horizontale Verschiebung (nach links ist negativ), die zweite Zahl bestimmt die vertikale Verschiebung (nach oben ist positiv). Nach einigen Experimenten wurden hier folgende Werte für gut befunden:

```
  % temporary code to break this example:
  \override TextScript #'outside-staff-priority = ##f
\once \override TextScript #'extra-offset = #'( -1.6 . 1.0 )
e4^\markup{ \italic ritenuto } g b e
```



Auch diese Zahlen sind nur Resultat einigen Herumprobierens und Vergleichens der Ergebnisse. Sie wollen den Text vielleicht etwas höher oder etwas mehr nach links setzen. Versuchen Sie es selber und vergleichen Sie das Ergebnis.

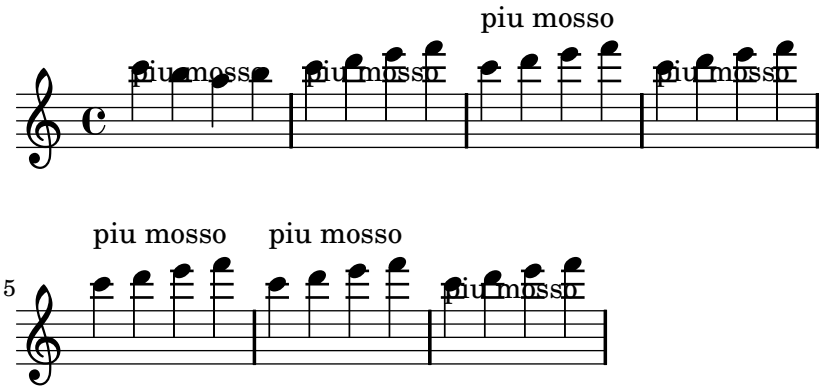
Eine letzte Warnung: in diesem Kapitel haben wir den Befehl

```
\once \override TextScript ...
```

benutzt. Dieser Befehl verändert die Anzeige des Textes für die nächste Note. Wenn die Note keinen Text zugeordnet hat, wird auch nichts verändert (und es wird **nicht** nach dem nächsten Text gesucht). Um das Verhalten zu verändern, so dass alles, was nach dem Befehl kommt, verändert wird, müssen Sie den Befehl `\once` weglassen. Um die Veränderung zu stoppen, benutzen Sie den Befehl `\revert`. Das wird genauer im Kapitel Abschnitt “The \override command” in *Benutzerhandbuch* erklärt.

```
  % temporary code to break this example:
  \override TextScript #'outside-staff-priority = ##f
c4^"piu mosso" b
\once \override TextScript #'padding = #4.6
  a4 b
c4^"piu mosso" d e f
\once \override TextScript #'padding = #4.6
c4^"piu mosso" d e f
c4^"piu mosso" d e f
\break
\override TextScript #'padding = #4.6
```

```
c4^"piu mosso" d e f
c4^"piu mosso" d e f
\revert TextScript #'padding
c4^"piu mosso" d e f
```



Siehe auch

Abschnitt “The \override command” in *Benutzerhandbuch*, Abschnitt 4.6 [Übliche Optimierungen], Seite 72.

4.5.2 Überlappende Notation in Ordnung bringen

Im Kapitel Abschnitt 4.5.1 [Verschieben von Objekten], Seite 68 wurde gezeigt, wie man Texte (`TextScript`-Objekte) verschiebt. Mit der gleichen Technik können auch andere Objektklassen verschoben werden, `TextScript` muss dann nur durch den Namen des Objektes ersetzt werden.

Um den Objektnamen zu finden, siehe die ‚**see also**‘-Hinweise am Ende des jeweiligen Abschnittes. Zum Beispiel am Ende des Kapitels Abschnitt “Dynamik” in *Benutzerhandbuch* findet sich:

Siehe auch

Programmreferenz: Abschnitt “DynamicText” in *Programmreferenz*, Abschnitt “Hairpin” in *Programmreferenz*. Vertikale Positionierung dieser Symbole wird mit Abschnitt “DynamicLineSpanner” in *Programmreferenz* erreicht.

Um also Dynamik-Zeichen zu verschieben, muss

```
\override DynamicLineSpanner #'padding = #2.0
```

benutzt werden. Es ist nicht genügend Platz, um jedes Objekt aufzulisten, aber die gebräuchlichsten finden sich hier:

Objekttyp	**Objektbezeichnung**
Dynamikzeichen (vertikal)	`DynamicLineSpanner`
Dynamikzeichen (horizontal)	`DynamicText`
Bindebögen	`Tie`
Phrasierungsbögen	`Slur`
Artikulationszeichen	`Script`
Fingersatz	`Fingering`
Text, z. B. `^"text"`	`TextScript`
Übungs-/Textmarken	`RehearsalMark`

4.5.3 Beispiele aus dem Leben

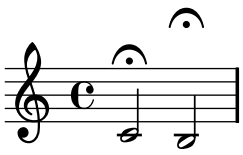
4.6 Übliche Optimierungen

Bestimmte Korrekturen sind so häufig, dass für sie schon fertige angepasste Befehle bereitgestellt sind, so etwa `\slurUp` um einen Bindebogen oberhalb anzuzeigen oder `\stemDown` um den Notenhals nach unten zu zwingen. Diese Befehle sind im Teil Alles über die Notation unter dem entsprechenden Abschnitt erklärt.

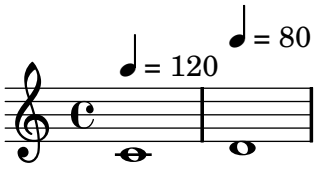
Eine vollständige Liste aller Veränderungen, die für jeden Objekttypen (etwa Bögen oder Balken) zur Verfügung stehen, ist in der Programmreferenz dargestellt. Viele Layoutobjekte benutzen jedoch gleiche Eigenschaften, die benutzt werden können, um eigene Einstellungen vorzunehmen.

- `padding`-Eigenschaft kann gesetzt werden, um den Abstand zwischen Symbolen über oder unter den Noten zu vergrößern oder zu verkleinern. Das gilt für alle Objekte, die ein `side-position-interface` besitzen, also unterscheiden, auf welcher Seite der Note sie sich befinden.

```
c2\fermata
\override Script #'padding = #3
b2\fermata
```



```
% Das funktioniert nicht, siehe unten
\override MetronomeMark #'padding = #3
\tempo 4=120
c1
% Das funktioniert:
\override Score.MetronomeMark #'padding = #3
\tempo 4=80
d1
```

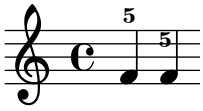


Im zweiten Beispiel ist es sehr wichtig zu wissen, welcher Kontext für bestimmte Objekte zuständig ist. Weil das `MetronomeMark`-Objekt vom `Score`-Kontext gesetzt wird, werden Veränderungen innerhalb des `Voice`-Kontextes nicht berücksichtigt. Genauere Details im Kapitel Abschnitt “The \override command” in *Benutzerhandbuch*.

- Die `extra-offset`-Eigenschaft verschiebt Objekte, hier ist ein Zahlenpaar zur Angabe der Positionierung erforderlich. Die erste Nummer bestimmt die horizontale Bewegung, eine positive Zahl bewegt das Objekt nach rechts. Die zweite Zahl bestimmt die vertikale Bewegung, eine positive Zahl bewegt das Objekt nach oben. Die `extra-offset`-Eigenschaft läuft auf unterster Ebene ab: Die Formatierungsmaschine ist sich der Veränderungen nicht bewusst.

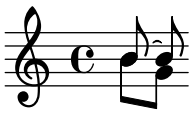
  Im folgenden Beispiel wird die zweite Fingersatzbezeichnung etwas nach links verschoben und 1,8 Notenzeilenabstände nach unten:

```
\stemUp
f-5
\once \override Fingering
    #'extra-offset = #'(-0.3 . -1.8)
f-5
```



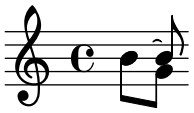
- Die Verwendung der `transparent`-Eigenschaft druckt das entsprechende Objekt mit „unsichtbarer Druckerschwärze“: Das Objekt wird nicht angezeigt, aber sein Verhalten bleibt bestehen. Das Objekt nimmt weiterhin Platz ein, es nimmt teil an Überschneidungen und deren Auflösung durch das Programm, Bögen und Balken können daran angebunden werden.

  Das nächste Beispiel zeigt, wie man unterschiedliche Stimmen mit Bindebögen verbinden kann. Normalerweise können Bindebögen nur zwei Noten der selben Stimme verbinden. Indem aber ein Bogen in einer anderen Stimme erstellt wird,



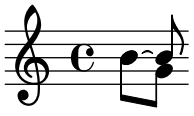
  und dann der erste Hals nach oben unsichtbar gemacht wird, scheint der Bindebogen die Stimme zu wechseln:

```
<< {
  \once \override Stem #'transparent = ##t
  b8~ b8\noBeam
} \\ {
  b[ g8]
} >>
```



  Damit der Hals den Bogen nicht zu sehr verkleinert, wird seine Länge (`length`) auf den Wert `8` gesetzt:

```
<< {
  \once \override Stem #'transparent = ##t
  \once \override Stem #'length = #8
  b8~ b8\noBeam
} \\ {
  b[ g8]
} >>
```



Abstände in LilyPond werden in Notenzeilenabständen (`staff-space`) gemessen, während die meisten Dicke-Eigenschaften auf mit der Notenliniendicke korrespondieren. Eine Eigenschaften verhalten sich anders, etwa die Dicke von Balken ist an die Notenzeilenabstände gekoppelt. Mehr Information findet sich im relevanten Teil der Programmreferenz.

4.7 Weitere Optimierungen

4.7.1 Andere Benutzung von Optimierungen

4.7.2 Variablen für Optimierungen einsetzen

4.7.3 Mehr Information

Die Programmreferenz enthält sehr viel Information über LilyPond, aber noch mehr Information findet sich in den internen LilyPond-Dateien.

Eine Standardeinstellungen (wie die Definitionen für den Kopf (`\header` sind als `.ly`-Datei gespeichert. Andere Einstellungen (wie die Definition für Beschriftung (`markup`) sind als `.scm` (Scheme)-Datei gespeichert. Eine nähere Erklärung geht über den Rahmen dieses Handbuches hinaus. Der Hinweis scheint aber angebracht, dass es grundlegende technische Kenntnis und sehr viel Zeit erfordert, diese Dateien zu verstehen.

- Linux: '*installdir*`/lilypond/usr/share/lilypond/current/`'
- OS X: '*installdir*`/LilyPond.app/Contents/Resources/share/lilypond/current/`'. Um diese Ordner anzuschauen, wechseln Sie entweder mit `cd` im Terminal zu der Adresse oder klicken Sie mit der rechten Maustaste auf das LilyPond-Symbol und wählen Sie ‚Show Package Contents'.
- Windows: '*installdir*`/LilyPond/usr/share/lilypond/current/`'

Die '`ly/`' und '`scm/`'-Ordner sind von besonderem Interesse. Dateien wie '`ly/property-init.ly`' und '`ly/declarations-init.ly`' definieren alle häufig vorkommenden Veränderungen.

4.7.4 Vermeiden von Optimierungen durch langsamere Übersetzung

LilyPond kann einige zusätzliche Tests durchführen, während die Noten gesetzt werden. Dadurch braucht das Programm länger, um den Notensatz zu produzieren, aber üblicherweise werden weniger nachträgliche Anpassungen nötig sein.

```
%%  Um sicher zu gehen, dass Texte und Liedtext
%%  innerhalb der Papierränder bleiben
\override Score.PaperColumn #'keep-inside-line = ##t
```

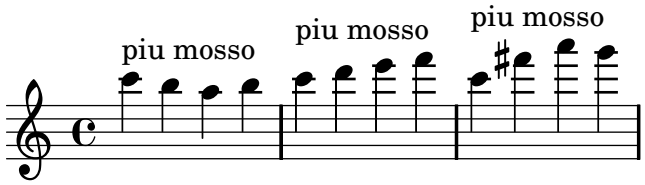
4.7.5 Fortgeschrittene Optimierungen mit Scheme

Es wurde schon gezeigt, wie die LilyPond-Ausgabe sehr stark verändert werden kann, indem man Befehle wie `\override TextScript #'extra-offset = ( 1 . -1)` benutzt. Aber noch mehr Einfluss auf die Formatierung kann durch den Einsatz von Scheme genommen werden. Eine vollständige Erklärung findet sich in der Anhang B [Scheme-Übung], Seite 118 und den Abschnitt "Schnittstellen für Programmierer" in *Benutzerhandbuch*.

Scheme kann benutzt werden, um einfach nur Befehle zu „überschreiben" (`\override`):

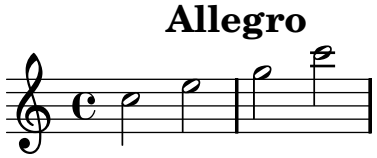
```
AbstandText = #(define-music-function (parser location padding) (number?)
#{
  \once \override TextScript #'padding = #$padding
#})
```

```
\relative c''' {
  c4^"piu mosso" b a b
  \AbstandText #1.8
  c4^"piu mosso" d e f
  \AbstandText #2.6
  c4^"piu mosso" fis a g
}
```



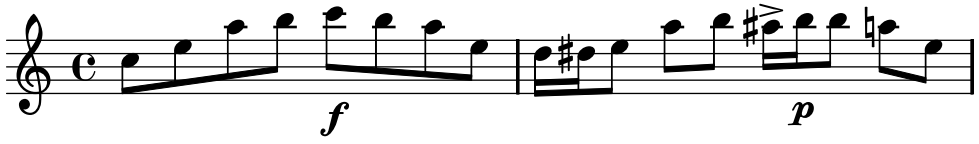
Hiermit können aber auch neue Befehle erstellt werden:

```
tempoZeichen = #(define-music-function (parser location padding marktext)
                                       (number? string?)
#{
  \once \override Score . RehearsalMark #'padding = $padding
  \once \override Score . RehearsalMark #'extra-spacing-width = #'(+inf.0 . -inf.0)
  \mark \markup { \bold $marktext }
#})

\relative c'' {
  c2 e
  \tempoZeichen #3.0 #"Allegro"
  g c
}
```



Sogar ganze musikalische Ausdrücke können eingefügt werden:

```
Muster = #(define-music-function (parser location x y) (ly:music? ly:music?)
#{
  $x e8 a b $y b a e
#})

\relative c''{
  \Muster c8 c8\f
  \Muster {d16 dis} { ais16-> b\p }
}
```



5 An LilyPond-Projekten arbeiten

Dieses Kapitel erklärt, wie bestimmte häufige Probleme zu lösen oder ganz zu vermeiden sind. Wenn Sie schon Programmiererfahrung mitbringen, erscheinen diese Hinweise vielleicht überflüssig, aber es wird dennoch empfohlen, dieses Kapitel zu lesen.

5.1 Vorschläge, wie LilyPond-Eingabe-Dateien geschrieben werden sollen

Jetzt sind Sie so weit, größere Stücke mit LilyPond zu schreiben – nicht nur die kleinen Beispiele aus der Übung, sondern ganze Stücke. Aber wie geht man das am besten an?

Solange LilyPond Ihre Dateien versteht und die Noten so setzt, wie Sie das wollen, spielt es eigentlich keine Rolle, wie Ihre Dateien aussehen. Es gibt aber trotzdem ein paar Dinge, die man beim Schreiben von LilyPond-Code berücksichtigen sollte.

- Was ist, wenn Sie einen Fehler machen? Die Struktur einer LilyPond-Datei kann es erleichtern (oder erschweren), bestimmte Fehler zu finden.
- Was ist, wenn Sie Ihre Dateien mit jemandem austauschen wollen? Oder Ihre Dateien nach einige Jahren noch einmal überarbeiten wollen? Manche LilyPond-Dateien versteht man auf den ersten Blick, über anderen muss man eine Stunde grübeln, um die Struktur zu ahnen.
- Was ist, wenn sie Ihre Dateien auf eine neuere LilyPond-Version aktualisieren wollen? Die Syntax der Eingabesprache verändert sich allmählich mit Verbesserungen im Programm. Die meisten Veränderungen können automatisch durch `convert-ly` gelöst werden, aber bestimmte Änderungen brauchen Handarbeit. LilyPond-Dateien können strukturiert werden, damit sie einfacher aktualisierbar sind.

5.1.1 Allgemeine Vorschläge

Hier einige Vorschläge, wie Sie Probleme vermeiden oder lösen können:

- **Schreiben Sie immer mit `\version` die Versionsnummer in jede Datei**. Beachten Sie, dass in allen Vorlagen die Versionsnummer `\version "2.11.51"` eingetragen ist. Es empfiehlt sich, in alle Dateien, unabhängig von ihrer Größe, den `\version`-Befehl einzufügen. Persönliche Erfahrung hat gezeigt, dass es ziemlich frustrierend sein kann zu erinnern, welche Programmversion man etwa vor einem Jahr verwendet hat. Auch `convert-ly` benötigt die Versionsnummer.
- **Benutzen Sie Überprüfungen**: Abschnitt “Oktavenüberprüfung” in *Benutzerhandbuch*, und Abschnitt “Takt- und Taktzahlüberprüfung” in *Benutzerhandbuch*. Wenn Sie hier und da diese Überprüfungen einfügen, finden Sie einen möglichen Fehler weit schneller. Wie oft aber ist „hier und da“? Das hängt von der Komplexität der Musik ab. Bei einfachen Stücken reicht es vielleicht ein- oder zweimal, in sehr komplexer Musik sollte man sie vielleicht in jeden Takt einfügen.
- **Ein Takt pro Textzeile**. Wenn irgendetwas kompliziertes vorkommt, entweder in der Musik selber oder in der Anpassung der Ausgabe, empfiehlt es sich oft, nur einen Takt pro Zeile zu schreiben. Bildschirmplatz zu sparen, indem Sie acht Takte in eine Zeile zwängen, hilft nicht weiter, wenn Sie ihre Datei „debuggen“ müssen.
- **Kommentieren Sie ihre Dateien**. Benutzen Sie entweder Taktnummern (in regelmäßigen Abständen) oder Verweise auf musikalische Themen („Zweites Thema in den Geigen“, „vierte Variation“ usw.). Sie brauchen diese Kommentare vielleicht noch nicht, wenn Sie das Stück notieren, aber spätestens wenn Sie nach ein paar Jahren etwas verändern wollen oder Sie den Quelltext an einen Freund weitergeben wollen, ist es weitaus komplizierter, die Dateistruktur ohne Kommentare zu verstehen, als wenn Sie sie rechtzeitig eingefügt hätten.

- **Schreiben Sie Klammern mit Einrückung**. Viele Probleme entstehen durch ungerade Anzahl von `{` and `}`-Klammern.
- **Schreiben Sie Tondauerangaben** am Anfang von Abschnitten und Bezeichnern. Wenn Sie beispielsweise `c4 d e` am Anfang eines Abschnittes schreiben, ersparen Sie sich viele Probleme, wenn Sie ihre Musik eines Tages umarrangieren wollen.
- **Trennen Sie Einstellungen** von den eigentlichen Noten. Siehe auch Abschnitt 5.1.4 [Tipparbeit sparen durch Bezeichner und Funktionen], Seite 78 und Abschnitt 5.1.5 [Stil-Dateien], Seite 79.

5.1.2 Das Kopieren von existierender Musik

Wenn Sie Musik aus einer fertigen Partitur kopieren (z. B. die LilyPond-Eingabe einer gedruckten Partitur):

- Schreiben Sie ein System ihrer Quelle nach dem anderen (aber trotzdem nur einen Takt pro Textzeile) und überprüfen Sie jedes System, nachdem Sie es fertig kopiert haben. Mit dem `showLastLength`-Befehl können Sie den Übersetzungsprozess beschleunigen. Siehe auch Abschnitt “Korrigierte Musik überspringen” in *Benutzerhandbuch*.
- Definieren Sie `mBreak = { \break }` und schreiben Sie `\mBreak` in der Quelldatei immer dann, wenn im Manuskript ein Zeilenumbruch vorkommt. Das macht es einfacher, die gesetzte Zeile mit den ursprünglichen Noten zu vergleichen. Wenn Sie die Partitur fertig gestellt haben, könne Sie `mBreak = { }`, also leer definieren, um diese manuellen Zeilenumbrüche zu entfernen. Damit kann dann LilyPond selber entscheiden, wohin es passende Zeilenumbrüche platziert.

5.1.3 Große Projekte

Besonders wenn Sie an größeren Projekten arbeiten, ist es unumgänglich, dass Sie ihre LilyPond-Dateien klar strukturieren.

- **Verwenden Sie Variablen für jede Stimme**, innerhalb der Definition sollte so wenig Struktur wie möglich sein. Die Struktur des `\score`-Abschnittes verändert sich am ehesten, während die `violin`-Definition sich wahrscheinlich mit einer neuen Programmversion nicht verändern wird.

```
violin = \relative c'' {
g4 c'8. e16
}
...
\score {
  \new GrandStaff {
    \new Staff {
      \violin
    }
  }
}
```

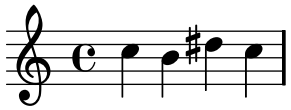
- **Trennen Sie Einstellungen von den Noten**. Diese Empfehlung wurde schon im Abschnitt Abschnitt 5.1.1 [Allgemeine Vorschläge], Seite 76 gegeben, aber für große Projekte ist es unumgänglich. Muss z. B. die Definition für `fdannp` verändert werden, so braucht man es nur einmal vorzunehmen und die Noten in der Geigenstimme, `violin`, bleiben unberührt.

```
fdannp = _\markup{
  \dynamic f \italic \small { 2nd } \hspace #0.1 \dynamic p }
violin = \relative c'' {
g4\fdannp c'8. e16
}
```

5.1.4 Tipparbeit sparen durch Bezeichner und Funktionen

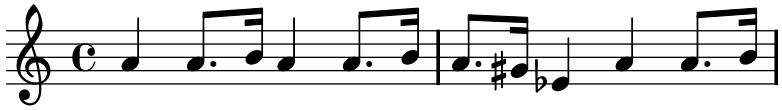
Bis jetzt haben Sie immer etwa solche Noten gesehen:

```
HornNoten = \relative c'' { c4 b dis c }
\score {
  {
    \HornNoten
  }
}
```



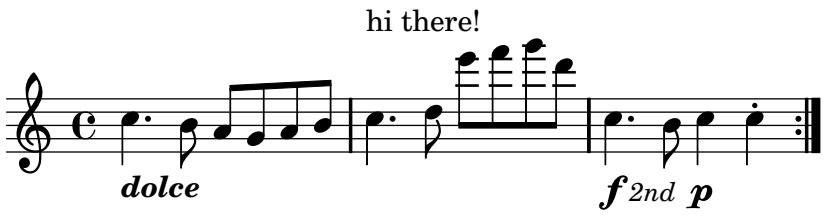
Das könnte auch nützlich in Minimal-Music sein:

```
fragA = \relative c'' { a4 a8. b16 }
fragB = \relative c'' { a8. gis16 ees4 }
Geige = \new Staff { \fragA \fragA \fragB \fragA }
\score {
  {
    \Geige
  }
}
```



Sie können diese Bezeichner oder Variablen aber auch für (eigene) Einstellungen verwenden:

```
dolce = \markup{ \italic \bold dolce }
AbstandText = { \once \override TextScript #'padding = #5.0 }
FdannP=_\markup{ \dynamic f \italic \small { 2nd } \hspace #0.1 \dynamic p }
Geige = \relative c'' {
  \repeat volta 2 {
    c4._\dolce b8 a8 g a b |
    \AbstandText
    c4.^"hi there!" d8 e' f g d |
    c,4.\FdannP b8 c4 c-. |
  }
}
\score {
  {
    \Geige
  }
\layout{ragged-right=##t}
}
```



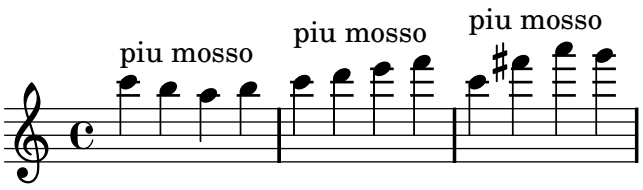
Die Variablen haben in diesem Beispiel deutlich die Tipparbeit erleichtert. Aber es lohnt sich, sie zu einzusetzen, auch wenn man sie nur einmal anwendet, denn sie vereinfachen die Struktur. Hier ist das vorangegangene Beispiel ohne Variablen. Es ist sehr viel komplizierter zu lesen, besonders die letzte Zeile.

```
violin = \relative c'' {
  \repeat volta 2 {
    c4._\markup{ \italic \bold dolce } b8 a8 g a b |
    \once \override TextScript #'padding = #5.0
    c4.^"hi there!" d8 e' f g d |
    c,4.\markup{ \dynamic f \italic \small { 2nd }
      \hspace #0.1 \dynamic p } b8 c4 c-. |
  }
}
```

Bis jetzt wurde nur statische Substitution vorgestellt – wenn LilyPond den Befehl `\padText` findet, wird er ersetzt durch durch unsere vorherige Definition (alles, was nach dem `padtext =` kommt).

LilyPond kennt aber auch nicht-statische Substitutionen (man kann sie sich als Funktionen vorstellen).

```
AbstandText =
#(define-music-function (parser location padding) (number?)
  #{
    \once \override TextScript #'padding = #$padding
  #})

\relative c''' {
  c4^"piu mosso" b a b
  \AbstandText #1.8
  c4^"piu mosso" d e f
  \AbstandText #2.6
  c4^"piu mosso" fis a g
}
```



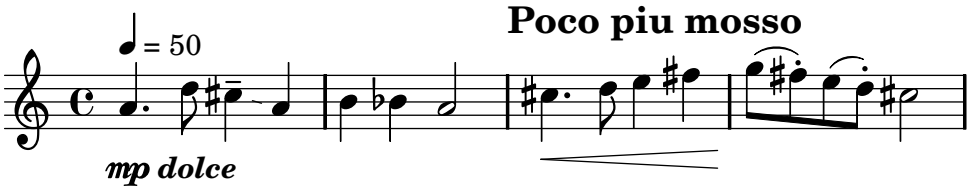
Die Benutzung von Variablen hilft auch, viele Schreibarbeit zu vermeiden, wenn die Eingabesyntax von LilyPond sich verändert (siehe auch Abschnitt 5.2.1 [Alte Dateien aktualisieren], Seite 83). Wenn nur eine einzige Definition (etwa `\dolce`) für alle Dateien verwendet wird (vgl. Abschnitt 5.1.5 [Stil-Dateien], Seite 79), muss nur diese einzige Definition verändert werden, wenn sich die Syntax ändert. Alle Verwendungen des Befehles beziehen sich dann auf die neue Definition.

5.1.5 Stil-Dateien

Die Ausgabe, die LilyPond erstellt, kann sehr stark modifiziert werden, siehe Kapitel 4 [Die Ausgabe verändern], Seite 68 für Einzelheiten. Aber wie kann man diese Änderungen auf eine ganze Serie von Dateien anwenden? Oder die Einstellungen von den Noten trennen? Das Verfahren ist ziemlich einfach.

Hier ist ein Beispiel. Es ist nicht schlimm, wenn Sie nicht auf Anhieb die Abschnitte mit den ganzen `#()` verstehen. Das wird im Kapitel Abschnitt 4.7.5 [Fortgeschrittene Optimierungen mit Scheme], Seite 74 erklärt.

```
mpdolce = #(make-dynamic-script (markup #:hspace 1 #:translate (cons 5 0)
  #:line(#:dynamic "mp" #:text #:italic "dolce" )))
tempoZeichen = #(define-music-function (parser location markp) (string?)
#{
  \once \override Score . RehearsalMark #'self-alignment-X = #left
  \once \override Score . RehearsalMark #'extra-spacing-width = #'(+inf.0 . -inf.0)
  \mark \markup { \bold $markp }
#})

\relative c'' {
  \tempo 4=50
  a4.\mpdolce d8 cis4--\glissando a | b4 bes a2
  \tempoZeichen "Poco piu mosso"
  cis4.\< d8 e4 fis | g8(\! fis)-. e( d)-. cis2
}
```



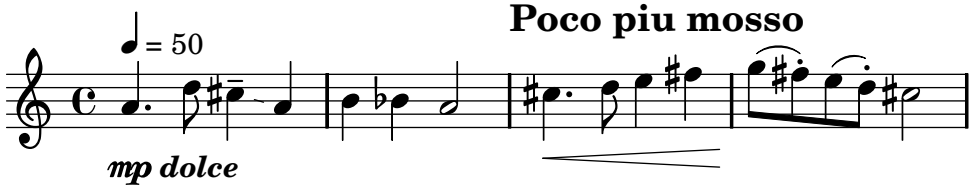
Es treten einige Probleme mit überlappenden Symbolen auf. Sie werden beseitigt mit den Tricks aus dem Kapitel Abschnitt 4.5.1 [Verschieben von Objekten], Seite 68. Aber auch die `mpdolce` und `tempoMark`-Definitionen können verbessert werden. Sie produzieren das Ergebnis, das gewünscht ist, aber es wäre schön, sie auch in anderen Stücken verwenden zu können. Man könnte sie natürlich einfach kopieren und in die anderen Dateien einfügen, aber das ist lästig. Die Definitionen verbleiben auch in der Notendatei und diese `#()` sehen nicht wirklich schön aus. Sie sollen in einer anderen Datei versteckt werden:

```
%%% speichern in einer Datei "definitions.ly"
mpdolce = #(make-dynamic-script (markup #:hspace 1 #:translate (cons 5 0)
  #:line(#:dynamic "mp" #:text #:italic "dolce" )))
tempoMark = #(define-music-function (parser location markp) (string?)
#{
  \once \override Score . RehearsalMark #'self-alignment-X = #left
  \once \override Score . RehearsalMark #'extra-spacing-width = #'(+inf.0 . -inf.0)
  \mark \markup { \bold $markp }
#})
```

Jetzt muss natürlich noch die Notendatei angepasst werden (gespeichert unter dem Namen ‘`"music.ly"`’).

```
\include "definitions.ly"

\relative c'' {
  \tempo 4=50
  a4.\mpdolce d8 cis4--\glissando a | b4 bes a2
  \once \override Score.RehearsalMark #'padding = #2.0
  \tempoMark "Poco piu mosso"
  cis4.\< d8 e4 fis | g8(\! fis)-. e( d)-. cis2
```

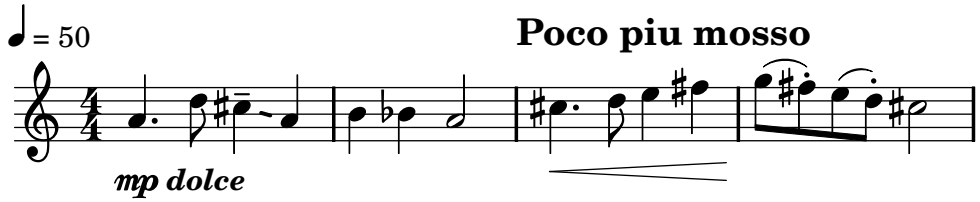
```
}
```



Das sieht schon besser aus, aber es sind noch einige Verbesserungen möglich. Das Glissando ist schwer zu sehen, also soll es etwas dicker erscheinen und dichter an den Notenköpfen gesetzt werden. Das Metronom-Zeichen soll über dem Schlüssel erscheinen, nicht über der ersten Note. Und schließlich kann unser Kompositionsprofessor „C“-Taktangaben überhaupt nicht leiden, also müssen sie in „4/4“ verändert werden.

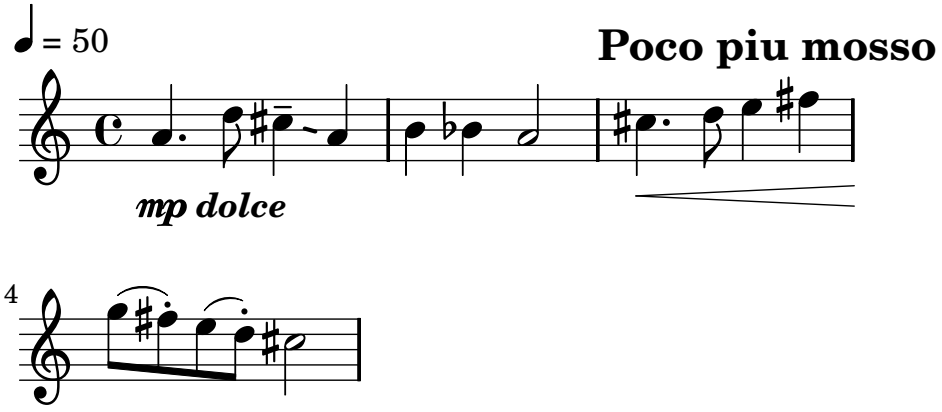
Diese Veränderungen sollten Sie aber nicht in der ‘`music.ly`’-Datei vornehmen. Ersetzen Sie die ‘`definitions.ly`’-Datei hiermit:

```
%%%  definitions.ly
mpdolce = #(make-dynamic-script (markup #:hspace 1 #:translate (cons 5 0)
  #:line( #:dynamic "mp" #:text #:italic "dolce" )))
tempoMark = #(define-music-function (parser location markp) (string?)
#{
  \once \override Score . RehearsalMark #'self-alignment-X = #left
  \once \override Score . RehearsalMark #'extra-spacing-width = #'(+inf.0 . -inf.0)
  \mark \markup { \bold $markp }
#})

\layout{
  \context { \Score
    \override MetronomeMark #'extra-offset = #'(-9 . 0)
    \override MetronomeMark #'padding = #'3
  }
  \context { \Staff
    \override TimeSignature #'style = #'numbered
  }
  \context { \Voice
    \override Glissando #'thickness = #3
    \override Glissando #'gap = #0.1
  }
}
```



Das sieht schon besser aus! Aber angenommen Sie möchten dieses Stück jetzt veröffentlichen. Ihr Kompositionsprofessor mag die „C“-Taktangaben nicht, aber Sie finden sie irgendwie schöner. Also kopieren Sie die Datei ‘`definitions.ly`’ nach ‘`web-publish.ly`’ und verändern diese. Weil die Noten in einer PDF-Datei auf dem Bildschirm angezeigt werden sollen, bietet es sich auch an, die gesamte Ausgabe zu vergrößern.

```
%%%  definitions.ly
mpdolce = #(make-dynamic-script (markup #:hspace 1 #:translate (cons 5 0)
  #:line( #:dynamic "mp" #:text #:italic "dolce" )))
tempoMark = #(define-music-function (parser location markp) (string?)
#{
  \once \override Score . RehearsalMark #'self-alignment-X = #left
  \once \override Score . RehearsalMark #'extra-spacing-width = #'(+inf.0 . -inf.0)
  \mark \markup { \bold $markp }
#})

#(set-global-staff-size 23)
\layout{
  \context { \Score
    \override MetronomeMark #'extra-offset = #'(-9 . 0)
    \override MetronomeMark #'padding = #'3
  }
  \context { \Staff
  }
  \context { \Voice
    \override Glissando #'thickness = #3
    \override Glissando #'gap = #0.1
  }
}
```



In der Notendatei muss jetzt nur noch `\include "definitions.ly"` durch `\include "web-publish.ly"` ausgetauscht werden. Das könnte man natürlich noch weiter vereinfachen. Also eine Datei '`definitions.ly`', die nur die Definitionen von `mpdolce` und `tempoMark` enthält, eine Datei '`web-publish.ly`', die alle die Änderungen für den `\layout`-Abschnitt enthält und eine Datei '`university.ly`' für eine Ausgabe, die den Wünschen des Professors entspricht. Der Anfang der '`music.ly`'-Datei würde dann so aussehen:

```
\include "definitions.ly"

%%%  Nur eine der beiden Zeilen auskommentieren!
\include "web-publish.ly"
%\include "university.ly"
```

Durch diese Herangehensweise kann auch bei der Erstellung von nur einer Ausgabeversion Arbeit gespart werden. Ich benutze ein halbes Dutzend verschiedener Stilvorlagen für meine Projekte. Jede Notationsdatei fängt an mit `\include "../global.ly"`, welches folgenden Inhalt hat:

```
%%%   global.ly
\version "2.11.51"
#(ly:set-option 'point-and-click #f)
\include "../init/init-defs.ly"
\include "../init/init-layout.ly"
\include "../init/init-headers.ly"
\include "../init/init-paper.ly"
```

5.2 Wenn etwas nicht funktioniert

5.2.1 Alte Dateien aktualisieren

Die Syntax von LilyPond verändert sich ab und zu. Wenn LilyPond besser wird, muss auch die Syntax (Eingabesprache) entsprechend angepasst werden. Teilweise machen diese Veränderungen die Eingabesprache einfacher lesbar, teilweise dienen sie dazu, neue Eigenschaften des Programmes benutzbar zu machen.

LilyPond stellt ein Programm bereit, das Aktualisierungen vereinfacht: `convert-ly`. Einzelheiten zur Programmbenutzung finden sich in Abschnitt “Dateien mit convert-ly aktualisieren” in *Programmbenutzung*.

Leider kann `convert-ly` nicht alle Veränderungen der Syntax berücksichtigen. Hier werden einfache „Suchen und Ersetzen“-Veränderungen vorgenommen (wie etwa `raggedright` zu becoming `ragged-right`), aber einige Veränderungen sind zu kompliziert. Die Syntax-Veränderungen, die das Programm nicht berücksichtigt, sind im Kapitel Abschnitt “Dateien mit convert-ly aktualisieren” in *Programmbenutzung* aufgelistet.

Zum Beispiel wurden in LilyPond 2.4 und früheren Versionen Akzente und Umlaute mit LaTeX-Befehlen eingegeben, ein „No\"el“ etwa ergäbe das französische Wort für Weihnachten. In LilyPond 2.6 und höher müssen diese Sonderzeichen direkt als utf-8-Zeichen eingegeben werden, in diesem Fall also „ë“. `convert-ly` kann nicht alle dieser LaTeX-Befehle verändern, das muss manuell vorgenommen werden.

5.2.2 Fehlersuche (alles auseinandernehmen)

Früher oder später werden Sie in die Lage kommen, dass LilyPond Ihre Datei nicht kompilieren will. Die Information, die LilyPond während der Übersetzung gibt, können Ihnen helfen, den Fehler zu finden, aber in vielen Fällen müssen Sie nach der Fehlerquelle auf die Suche gehen.

Die besten Hilfsmittel sind in diesem Fall das Zeilen- und Blockkommentar (angezeigt durch `%` bzw. `%{ ... %}`). Wenn Sie nicht bestimmen können, wo sich das Problem befindet, beginnen Sie damit, große Teile des Quelltextes auszukommentieren. Nachdem Sie einen Teil auskommentiert haben, versuchen Sie, die Datei erneut zu übersetzen. Wenn es jetzt funktioniert, muss sich das Problem innerhalb der Kommentare befinden. Wenn es nicht funktioniert, müssen Sie weitere Teile auskommentieren bis sie eine Version haben, die funktioniert.

In Extremfällen bleibt nur noch solch ein Beispiel übrig:

```
\score {
  <<
    % \melody
    % \harmony
    % \bass
  >>
  \layout{}
}
```

(also eine Datei ohne Noten).

Geben Sie nicht auf, wenn das vorkommen sollte. Nehmen Sie das Kommentarzeichen von einem Teil wieder weg, sagen wir der Bassstimme, und schauen Sie, ob es funktioniert. Wenn nicht, dann kommentieren Sie die gesamte Bassstimme aus, aber nicht den `\bass`-Befehl in dem `\score`-Abschnitt:

```
bass = \relative c' {
%{
  c4 c c c
  d d d d
%}
}
```

Jetzt beginnen Sie damit, langsam Stück für Stück der Bassstimme wieder hineinzunehmen, bis Sie die problematische Zeile finden.

Eine andere nützliche Technik zur Problemlösung ist es, Abschnitt 5.2.3 [Minimalbeispiele], Seite 84 zu konstruieren.

5.2.3 Minimalbeispiele

Ein Minimalbeispiel ist eine Beispieldatei, die so klein wie möglich ist. Diese Beispiele sind sehr viel einfacher zu verstehen als die langen Originaldateien. Minimalbeispiele werden benutzt, um

- Fehlerberichte zu erstellen,
- eine Hilfeanfrage an die E-Mail-Liste zu schicken,
- Ein Beispiel zur LilyPond Schnipselsammlunghinzuzufügen.

Um ein Beispiel zu konstruieren, das so klein wie möglich ist, gibt es eine einfache Regel: Alles nicht Notwendige entfernen. Wenn Sie unnötige Teile einer Datei entfernen, bietet es sich an, sie auszukommentieren und nicht gleich zu löschen. Auf diese Weise können Sie eine Zeile leicht wieder mit aufnehmen, sollten Sie sie doch brauchen, anstatt sie von Anfang an neu zu schreiben.

Es gibt zwei Ausnahmen dieser „So klein wie möglich“-Regel:

- Fügen Sie immer einen `\version`Befehl ein.
- Wenn es möglich ist, benutzen Sie `\paper{ ragged-right = ##t }` am Beginn des Beispiels.

Der Sinn der Minimalbeispiele ist, dass sie einfach lesbar sind:

- Vermeiden Sie es, komplizierte Noten, Schlüssel oder Taktangaben zu verwenden, es sei denn, Sie wollen genau an diesen Elementen etwas demonstrieren.
- Benutzen Sie keine `\override`-Befehle, wenn sie nicht der Zweck des Beispieles sind.

5.3 Partituren und Stimmen

Orchesternoten werden alle zweimal gesetzt. Erstens als Stimmen für die Musiker, und dann als große Partitur für den Dirigenten. Mit Variablen kann hier doppelte Arbeit erspart werden. Die Musik muss nur einmal eingegeben werden und wird in einer Variable abgelegt. Der Inhalt dieser Variable wird dann benutzt, um sowohl die Stimme als auch die Partitur zu erstellen.

Es bietet sich an, die Noten in eigenen Dateien zu speichern. Sagen wir beispielsweise, dass in der Datei ‘`Horn-Noten.ly`’ die folgenden Noten eines Duetts für Horn und Fagott gespeichert sind:

```
HornNoten = \relative c {
  \time 2/4
  r4 f8 a cis4 f e d
}
```

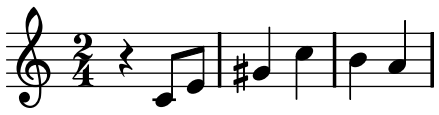
Daraus wird dann eine eigene Stimme gemacht, indem folgende Datei erstellt wird:

```
\include "Horn-Noten.ly"
\header {
  instrument = "Horn in F"
}

{
 \transpose f c' \HornNoten
}
```

Die Zeile

```
\include "Horn-Noten.ly"
```

setzt den Inhalt der Datei '`Horn-Noten.ly`' an die Stelle des Befehls in die aktuelle Datei. Damit besteht also eine Definition für `HornNoten`, so dass die Variable verwendet werden kann. Der Befehl `\transpose f c'` zeigt an, dass das Argument, also `\HornNoten`, um eine Quinte nach oben transponiert wird. Klingendes ‚f' wird also als `c'` notiert. Das entspricht der Notation eines Waldhorn in F. Die Transposition zeigt die folgende Ausgabe:



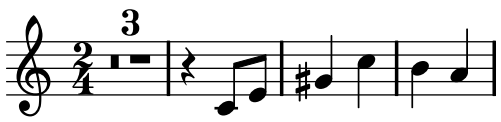
In Musik für mehrere Instrumente kommt es oft vor, dass eine Stimme für mehrere Takte nicht spielt. Das wird mit einer besonderen Pause angezeigt, dem Pausenzeichen für mehrere Takte (engl. multi-measure rest). Sie wird mit dem *großen* Buchstaben '`R`' eingegeben, gefolgt von einer Dauer (`1` für eine Ganze, `2` für eine Halbe usw.). Indem man die Dauer multipliziert, können längere Pausen erstellt werden. Z. B. dauert diese Pause drei Takte eines 2/4-Taktes:

```
R2*3
```

Wenn die Stimme gedruckt wird, müssen diese Pausen zusammengezogen werden. Das wird durch eine Variable erreicht:

```
\set Score.skipBars = ##t
```

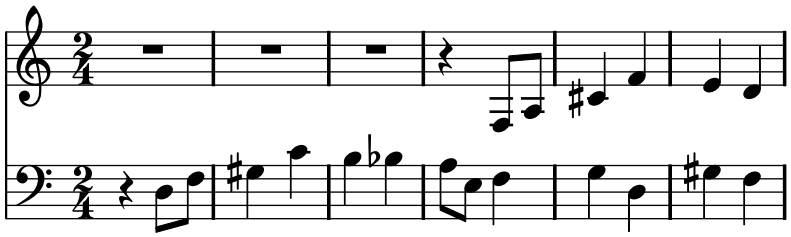
Dieser Befehl setzt die Eigenschaft des `skipBars` („überspringe Takte") auf wahr (`##t`). Wenn diese Option und die Pause zu der Musik des Beispiels gesetzt wird, erhält man folgendes Ergebnis:



Die Partitur wird erstellt, indem alle Noten zusammengesetzt werden. Angenommen, die andere Stimme trägt den Namen `FagottNoten` und ist in der Datei '`Fagott-Noten.ly`' gespeichert. Die Partitur sieht dann folgendermaßen aus:

```
\include "Fagott-Noten.ly"
\include "Horn-Noten.ly"

<<
  \new Staff \HornNoten
  \new Staff \FagottNoten
>>
```

Und mit LilyPond übersetzt:



Tiefer gehende Information darüber, wie Stimmauszüge und Partituren erstellt werden, finden sich im Notationshandbuch, siehe Abschnitt 5.3 [Partituren und Stimmen], Seite 84.

Das Setzen der Variablen, die das Verhalten von LilyPond beeinflussen (‚properties‘), wird im Kapitel Abschnitt 3.3.4 [Umgebungs-Eignschaften verändern], Seite 67 besprochen.

Anhang A Vorlagen

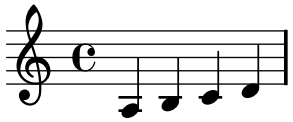
Dieser Abschnitt des Handbuches enthält Vorlagen, in denen die LilyPond-Partitur schon eingerichtet ist. Sie müssen nur noch Ihre Noten einfügen, die Datei mit LilyPond übersetzen und sich an dem schönen Notenbild erfreuen!

A.1 Ein einzelnes System

A.1.1 Nur Noten

Das erste Beispiel zeigt ein Notensystem mit Noten, passend für ein Soloinstrument oder ein Melodiefragment. Kopieren Sie es und fügen Sie es in Ihre Datei ein, schreiben Sie die Noten hinzu, und Sie haben eine vollständige Notationsdatei.

```
\version "2.11.51"
Melodie = \relative c' {
   \clef treble
   \key c \major
   \time 4/4

   a4 b c d
}

\score {
   \new Staff \Melodie
   \layout { }
   \midi {}
}
```

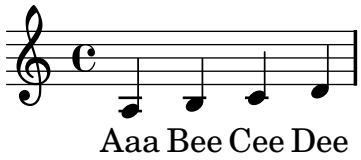


A.1.2 Noten und Text

Das nächste Beispiel zeigt eine einfache Melodie mit Text. Kopieren Sie es in Ihre Datei, fügen Sie Noten und Text hinzu und übersetzen Sie es mit LilyPond. In dem Beispiel wird die automatische Balkenverbindung ausgeschaltet (mit dem Befehl `\autoBeamOff`), wie es für Vokalmusik üblich ist. Wenn Sie die Balken wieder einschalten wollen, müssen Sie die entsprechende Zeile entweder ändern oder auskommentieren.

```
\version "2.11.51"
Melodie = \relative c' {
   \clef treble
   \key c \major
   \time 4/4

   a4 b c d
}

Text = \lyricmode {
   Aaa Bee Cee Dee
}
```

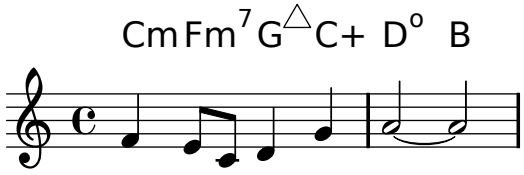
```
\score{
   <<
      \new Voice = "one" {
         \autoBeamOff
         \Melodie
      }
      \new Lyrics \lyricsto "one" \Text
   >>
   \layout { }
   \midi { }
}
```



A.1.3 Noten und Akkordbezeichnungen

Wollen Sie ein Liedblatt mit Melodie und Akkorden schreiben? Hier ist das richtige Beispiel für Sie!

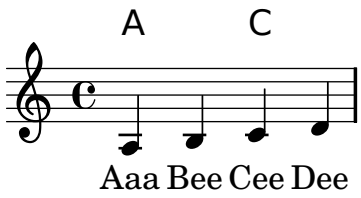
```
\version "2.11.51"
Melodie = \relative c' {
   \clef treble
   \key c \major
   \time 4/4

   f4 e8[ c] d4 g |
   a2 ~ a2 |
}

harmonies = \chordmode {
   c4:m f:min7 g:maj c:aug d2:dim b:sus
}

\score {
   <<
      \new ChordNames {
         \set chordChanges = ##t
         \harmonies
      }
   \new Staff \Melodie
   >>

   \layout{ }
   \midi { }
}
```



A.1.4 Noten, Text und Akkordbezeichnungen

Mit diesem Beispiel können Sie einen Song mit Melodie, Text und Akkorden schreiben.

```
\version "2.11.51"
Melodie = \relative c' {
   \clef treble
   \key c \major
   \time 4/4

   a b c d
}

Text = \lyricmode {
   Aaa Bee Cee Dee
}

harmonies = \chordmode {
   a2 c2
}

\score {
   <<
      \new ChordNames {
         \set chordChanges = ##t
         \harmonies
      }
   \new Voice = "one" {
      \autoBeamOff
      \Melodie
   }
   \new Lyrics \lyricsto "one" \Text
   >>
   \layout { }
   \midi { }
}
```



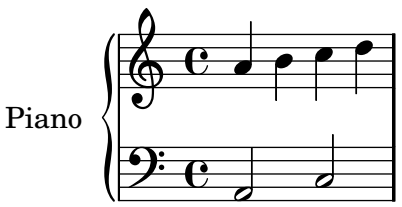
A.2 Klaviervorlagen

A.2.1 Piano Solo

Hier kommt ein einfaches Klaviersystem.

```
\version "2.11.51"
```

```
oben = \relative c'' {
   \clef treble
   \key c \major
   \time 4/4

   a b c d
}

unten = \relative c {
   \clef bass
   \key c \major
   \time 4/4

   a2 c
}

\score {
   \new PianoStaff <<
      \set PianoStaff.instrumentName = "Piano  "
      \new Staff = "upper" \oben
      \new Staff = "lower" \unten
   >>
   \layout { }
   \midi { }
}
```

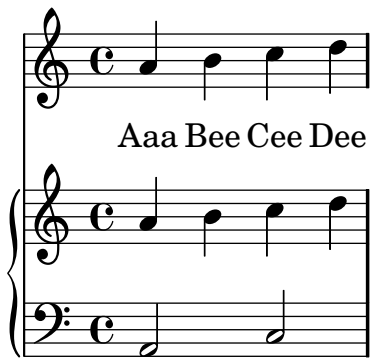


A.2.2 Klavier und Gesangstimme

Das nächste Beispiel ist typisch für ein Lied: Im oberen System die Melodie mit Text, darunter Klavierbegleitung.

```
\version "2.11.51"
Melodie = \relative c'' {
   \clef treble
   \key c \major
   \time 4/4

   a b c d
}

Text = \lyricmode {
   Aaa Bee Cee Dee
}

oben = \relative c'' {
   \clef treble
```

```
   \key c \major
   \time 4/4

   a b c d
}

unten = \relative c {
   \clef bass
   \key c \major
   \time 4/4

   a2 c
}

\score {
   <<
      \new Voice = "mel" {
          \autoBeamOff
          \Melodie
      }
      \new Lyrics \lyricsto mel \Text

      \new PianoStaff <<
         \new Staff = "upper" \oben
         \new Staff = "lower" \unten
      >>
   >>
   \layout {
      \context { \RemoveEmptyStaffContext }
   }
   \midi { }
}
```

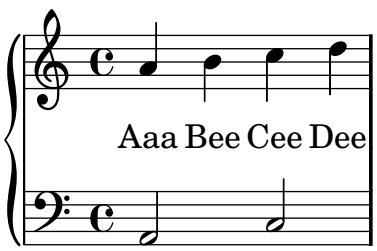


A.2.3 Klavier mit zentriertem Text

Anstatt ein eigenes System für Melodie und Text zu schreiben, können Sie den Text auch zwischen die beiden Klaviersysteme schreiben (und damit das zusätzliche System für die Gesangstimme auslassen).

```
\version "2.11.51"
oben = \relative c'' {
```

```
   \clef treble
   \key c \major
   \time 4/4

   a b c d
}

unten = \relative c {
   \clef bass
   \key c \major
   \time 4/4

   a2 c
}

Text = \lyricmode {
   Aaa Bee Cee Dee
}

\score {
  \new GrandStaff <<
    \new Staff = oben { \new Voice = "singer" \oben }
    \new Lyrics \lyricsto "singer" \Text
    \new Staff = unten {
      \clef bass
      \unten
    }
  >>
  \layout {
    \context { \GrandStaff \accepts "Lyrics" }
    \context { \Lyrics \consists "Bar_engraver" }
  }
  \midi { }
}
```



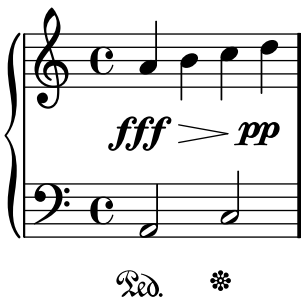
A.2.4 Klavier mit zentrierten Lautstärkebezeichnungen

In der meisten Klaviernotation werden die Dynamikzeichen zwischen den beiden Systemen zentriert. Für LilyPond muss man die Einstellungen etwas anpassen, aber Sie können ja das angepasste Beispiel von hier kopieren.

```
\version "2.11.51"
oben = \relative c'' {
  \clef treble
  \key c \major
```

```
  \time 4/4

  a b c d
}

unten = \relative c {
  \clef bass
  \key c \major
  \time 4/4

  a2 c
}

Dynamik = {
  s2\fff\> s4
  s\!\pp
}

pedal = {
  s2\sustainOn s2\sustainOff
}

\score {
  \new PianoStaff <<
    \new Staff = "upper" \oben
    \new Dynamics = "dynamics" \Dynamik
    \new Staff = "lower" <<
      \clef bass
      \unten
    >>
    \new Dynamics = "pedal" \pedal
  >>
  \layout {
    \context {
      \type "Engraver_group"
      \name Dynamics
      \alias Voice % So that \cresc works, for example.
      \consists "Output_property_engraver"

      \override VerticalAxisGroup #'minimum-Y-extent = #'(-1 . 1)
      \override DynamicLineSpanner #'Y-offset = #0
      pedalSustainStrings = #'("Ped." "*Ped." "*")
      pedalUnaCordaStrings = #'("una corda" "" "tre corde")

      \consists "Piano_pedal_engraver"
      \consists "Script_engraver"
      \consists "Dynamic_engraver"
      \consists "Text_engraver"

      \override TextScript #'font-size = #2
      \override TextScript #'font-shape = #'italic
```

```
      \consists "Skip_event_swallow_translator"

      \consists "Axis_group_engraver"
    }
    \context {
      \PianoStaff
      \accepts Dynamics
    }
  }
}
\score {
  \new PianoStaff <<
    \new Staff = "upper" << \oben \Dynamik >>
    \new Staff = "lower" << \unten \Dynamik >>
    \new Dynamics = "pedal" \pedal
  >>
  \midi {
    \context {
      \type "Performer_group"
      \name Dynamics
      \consists "Piano_pedal_performer"
    }
    \context {
      \PianoStaff
      \accepts Dynamics
    }
  }
}
```



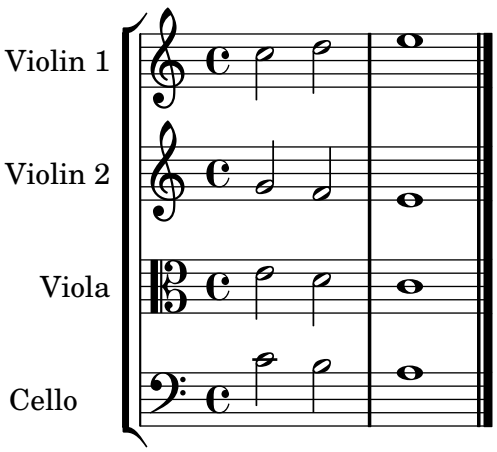
A.3 Streichquartett

A.3.1 Streichquartett

Dieses Beispiel demonstriert die Partitur für ein Streichquartett. Hier wird auch eine „`\global`“-Variable für Taktart und Vorzeichen benutzt.

```
\version "2.11.51"

global= {
  \time 4/4
  \key c \major
}
```

```
violinOne = \new Voice { \relative c''{
  \set Staff.instrumentName = "Violin 1 "

  c2 d e1

\bar "|." }}
violinTwo = \new Voice { \relative c''{
  \set Staff.instrumentName = "Violin 2 "

  g2 f e1

\bar "|." }}
viola = \new Voice { \relative c' {
  \set Staff.instrumentName = "Viola "
  \clef alto

  e2 d c1

\bar "|." }}
Cello = \new Voice { \relative c' {
  \set Staff.instrumentName = "Cello     "
  \clef bass

  c2 b a1

\bar "|."}}

\score {
   \new StaffGroup <<
      \new Staff << \global \violinOne >>
      \new Staff << \global \violinTwo >>
      \new Staff << \global \viola >>
      \new Staff << \global \Cello >>
   >>
   \layout { }
   \midi { }
}
```



A.3.2 Streichquartettstimmen

Mit diesem Beispiel können Sie ein schönes Streichquartett notieren, aber wie gehen Sie vor, wenn Sie Stimmen brauchen? Das Beispiel oben hat gezeigt, wie Sie mit Variablen einzelne Abschnitte getrennt voneinander notieren können. Im nächsten Beispiel wird nun gezeigt, wie Sie mit diesen Variablen einzelne Stimmen erstellen.

Sie müssen das Beispiel in einzelne Dateien aufteilen; die Dateinamen sind in den Kommentaren am Anfang jeder Datei enthalten. `piece.ly` enthält die Noten. Die anderen Dateien – `score.ly`, `vn1.ly`, `vn2.ly`, `vla.ly` und `vlc.ly` – erstellen daraus die entsprechenden Stimmen bzw. die Partitur (`score.ly`). Mit `\tag` wird den Stimmen ein Name zugewiesen, auf den zurückgegriffen werden kann.

```
%%%%% piece.ly
\version "2.11.51"

global= {
  \time 4/4
  \key c \major
}

Violinone = \new Voice { \relative c''{
  \set Staff.instrumentName = "Violin 1 "

  c2 d e1

\bar "|." }}   %*********************************
Violintwo = \new Voice { \relative c''{
  \set Staff.instrumentName = "Violin 2 "

  g2 f e1

\bar "|." }}   %*********************************
Viola = \new Voice { \relative c' {
  \set Staff.instrumentName = "Viola "
  \clef alto

  e2 d c1

\bar "|." }}   %*********************************
Cello = \new Voice { \relative c' {
  \set Staff.instrumentName = "Cello     "
  \clef bass

  c2 b a1

\bar "|."}}   %*********************************

music = {
  <<
    \tag #'score \tag #'vn1 \new Staff { << \global \Violinone >> }
    \tag #'score \tag #'vn2 \new Staff { << \global \Violintwo>> }
    \tag #'score \tag #'vla \new Staff { << \global \Viola>> }
    \tag #'score \tag #'vlc \new Staff { << \global \Cello>> }
```

```
  >>
}


%%%%% score.ly
\version "2.11.51"
\include "piece.ly"
#(set-global-staff-size 14)
\score {
  \new StaffGroup \keepWithTag #'score \music
  \layout { }
  \midi { }
}


%%%%% vn1.ly
\version "2.11.51"
\include "piece.ly"
\score {
  \keepWithTag #'vn1 \music
  \layout { }
}


%%%%% vn2.ly
\version "2.11.51"
\include "piece.ly"
\score {
  \keepWithTag #'vn2 \music
  \layout { }
}


%%%%% vla.ly
\version "2.11.51"
\include "piece.ly"
\score {
  \keepWithTag #'vla \music
  \layout { }
}


%%%%% vlc.ly
\version "2.11.51"
\include "piece.ly"
\score {
  \keepWithTag #'vlc \music
  \layout { }
}
```

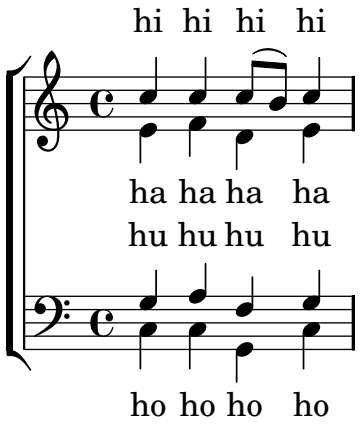
A.4 Vokalensemble

A.4.1 SATB-Partitur

Dieses Beispiel ist für vierstimmigen Gesang (SATB). Bei größeren Stücken ist es oft sinnvoll, eine allgemeine Variable zu bestimmen, die in allen Stimmen eingefügt wird. Taktart und Vorzeichen etwa sind fast immer gleich in allen Stimmen.

```
\version "2.11.51"
global = {
   \key c \major
   \time 4/4
}

sopMusic = \relative c'' {
   c4 c c8[( b)] c4
}
sopWords = \lyricmode {
   hi hi hi hi
}

AltNoten = \relative c' {
   e4 f d e
}
AltText =\lyricmode {
   ha ha ha ha
}

TenorNoten = \relative c' {
   g4 a f g
}
TenorText = \lyricmode {
   hu hu hu hu
}

BassNoten = \relative c {
   c4 c g c
}
BassText = \lyricmode {
   ho ho ho ho
}

\score {
   \new ChoirStaff <<
      \new Lyrics = sopranos { s1 }
      \new Staff = women <<
         \new Voice =
           "sopranos" { \voiceOne << \global \sopMusic >> }
         \new Voice =
           "altos" { \voiceTwo << \global \AltNoten >> }
      >>
      \new Lyrics = "altos" { s1 }
      \new Lyrics = "tenors" { s1 }
```

```
      \new Staff = men <<
         \clef bass
         \new Voice =
           "tenors" { \voiceOne <<\global \TenorNoten >> }
         \new Voice =
           "basses" { \voiceTwo <<\global \BassNoten >> }
      >>
      \new Lyrics = basses { s1 }

      \context Lyrics = sopranos \lyricsto sopranos \sopWords
      \context Lyrics = altos \lyricsto altos \AltText
      \context Lyrics = tenors \lyricsto tenors \TenorText
      \context Lyrics = basses \lyricsto basses \BassText
   >>

   \layout {
      \context {
         % a little smaller so lyrics
         % can be closer to the staff
         \Staff
         \override VerticalAxisGroup #'minimum-Y-extent = #'(-3 . 3)
      }
   }
}
```



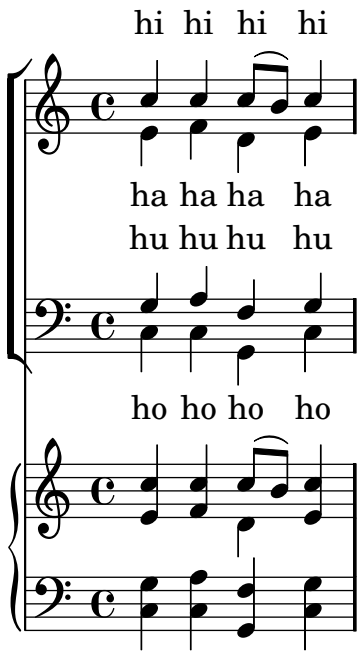
A.4.2 SATB-Partitur und automatischer Klavierauszug

In diesem Beispiel wird ein automatischer Klavierauszug zu der Chorpartitur hinzugefügt. Das zeigt eine der Stärken von LilyPond – man kann eine Variable mehr als einmal benutzen. Wenn Sie irgendeine Änderung an einer Chorstimme vornehmen, (etwa tenorMusic), verändert sich auch der Klavierauszug entsprechend.

```
\version "2.11.51"
global = {
   \key c \major
   \time 4/4
}

sopMusic = \relative c'' {
   c4 c c8[( b)] c4
```

```
}
sopWords = \lyricmode {
   hi hi hi hi
}

AltNoten = \relative c' {
   e4 f d e
}
AltText =\lyricmode {
   ha ha ha ha
}

TenorNoten = \relative c' {
   g4 a f g
}
TenorText = \lyricmode {
   hu hu hu hu
}

BassNoten = \relative c {
   c4 c g c
}
BassText = \lyricmode {
   ho ho ho ho
}

\score {
  <<
    \new ChoirStaff <<
      \new Lyrics = sopranos { s1 }
      \new Staff = women <<
        \new Voice =
          "sopranos" { \voiceOne << \global \sopMusic >> }
        \new Voice =
          "altos" { \voiceTwo << \global \AltNoten >> }
      >>
      \new Lyrics = "altos" { s1 }
      \new Lyrics = "tenors" { s1 }
      \new Staff = men <<
        \clef bass
        \new Voice =
          "tenors" { \voiceOne <<\global \TenorNoten >> }
        \new Voice =
          "basses" { \voiceTwo <<\global \BassNoten >> }
      >>
      \new Lyrics = basses { s1 }

      \context Lyrics = sopranos \lyricsto sopranos \sopWords
      \context Lyrics = altos \lyricsto altos \AltText
      \context Lyrics = tenors \lyricsto tenors \TenorText
      \context Lyrics = basses \lyricsto basses \BassText
    >>
```

```
  \new PianoStaff <<
    \new Staff <<
      \set Staff.printPartCombineTexts = ##f
      \partcombine
      << \global \sopMusic >>
      << \global \AltNoten >>
    >>
    \new Staff <<
      \clef bass
      \set Staff.printPartCombineTexts = ##f
      \partcombine
      << \global \TenorNoten >>
      << \global \BassNoten >>
    >>
   >>
  >>
  \layout {
    \context {
      % a little smaller so lyrics
      % can be closer to the staff
      \Staff
        \override VerticalAxisGroup #'minimum-Y-extent = #'(-3 . 3)
    }
  }
}
```



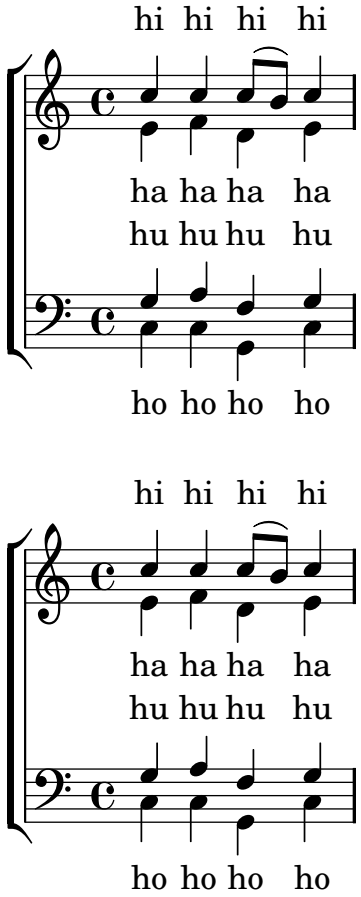
A.4.3 SATB mit zugehörigen Kontexten

In diesem Beispiel werden die Texte mit den Befehlen `alignAboveContext` und `alignBelowContext` über und unter dem System angeordnet.

```
\version "2.11.51"
global = {
  \key c \major
```

```
  \time 4/4
}

sopMusic = \relative c'' {
  c4 c c8[( b)] c4
}
sopWords = \lyricmode {
  hi hi hi hi
}

AltNoten = \relative c' {
  e4 f d e
}
AltText =\lyricmode {
  ha ha ha ha
}

TenorNoten = \relative c' {
  g4 a f g
}
TenorText = \lyricmode {
  hu hu hu hu
}

BassNoten = \relative c {
  c4 c g c
}
BassText = \lyricmode {
  ho ho ho ho
}

\score {
  \new ChoirStaff <<
     \new Staff = women <<
        \new Voice =
          "sopranos" { \voiceOne << \global \sopMusic >> }
        \new Voice =
          "altos" { \voiceTwo << \global \AltNoten >> }
     >>
     \new Lyrics \with {alignAboveContext=women} \lyricsto sopranos \sopWords
     \new Lyrics \with {alignBelowContext=women} \lyricsto altos \AltText
% we could remove the line about this with the line below, since we want
% the alto lyrics to be below the alto Voice anyway.
%    \new Lyrics \lyricsto altos \AltText

     \new Staff = men <<
        \clef bass
        \new Voice =
          "tenors" { \voiceOne <<\global \TenorNoten >> }
        \new Voice =
          "basses" { \voiceTwo <<\global \BassNoten >> }
     >>
```

```
      \new Lyrics \with {alignAboveContext=men} \lyricsto tenors \TenorText
      \new Lyrics \with {alignBelowContext=men} \lyricsto basses \BassText
% again, we could replace the line above this with the line below.
%    \new Lyrics \lyricsto basses \BassText
  >>

  \layout {
     \context {
        % a little smaller so lyrics
        % can be closer to the staff
        \Staff
        \override VerticalAxisGroup #'minimum-Y-extent = #'(-3 . 3)
     }
  }
}


\score {
  \new ChoirStaff <<
     \new Staff = women <<
        \new Voice =
          "sopranos" { \voiceOne << \global \sopMusic >> }
        \new Voice =
          "altos" { \voiceTwo << \global \AltNoten >> }
     >>

     \new Lyrics \with {alignAboveContext=women} \lyricsto sopranos \sopWords
     \new Lyrics \lyricsto altos \AltText

     \new Staff = men <<
        \clef bass
        \new Voice =
          "tenors" { \voiceOne <<\global \TenorNoten >> }
        \new Voice =
          "basses" { \voiceTwo <<\global \BassNoten >> }
     >>

     \new Lyrics \with {alignAboveContext=men} \lyricsto tenors \TenorText
     \new Lyrics \lyricsto basses \BassText
  >>

  \layout {
     \context {
        % a little smaller so lyrics
        % can be closer to the staff
        \Staff
        \override VerticalAxisGroup #'minimum-Y-extent = #'(-3 . 3)
     }
  }
}
```



A.5 Vorlagen für alte Notation

A.5.1 Transkription mensuraler Musik

Bei der Transkription von Mensuralmusik ist es oft erwünscht, ein Incipit an den Anfang des Stückes zu stellen, damit klar ist, wie Tempo und Schlüssel in der Originalnotation gesetzt waren. Während heutzutage Musiker an Taktlinien gewöhnt sind, um Rhythmen schneller zu erkennen, wurden diese in der Mensuralmusik nicht verwendet. Tatsächlich ändern sich die Rhythmen auch oft alle paar Noten. Als ein Kompromiss werden die Notenlinien nicht auf dem System, sondern zwischen den Systemen geschrieben.

```
\version "2.11.51"

global = {
  \set Score.skipBars = ##t

  % incipit
  \once \override Score.SystemStartBracket #'transparent = ##t
  \override Score.SpacingSpanner #'spacing-increment = #1.0 % tight spacing
  \key f \major
  \time 2/2
  \once \override Staff.TimeSignature #'style = #'neomensural
  \override Voice.NoteHead #'style = #'neomensural
  \override Voice.Rest #'style = #'neomensural
  \set Staff.printKeyCancellation = ##f
  \cadenzaOn % turn off bar lines
  \skip 1*10
  \once \override Staff.BarLine #'transparent = ##f
  \bar "||"
```

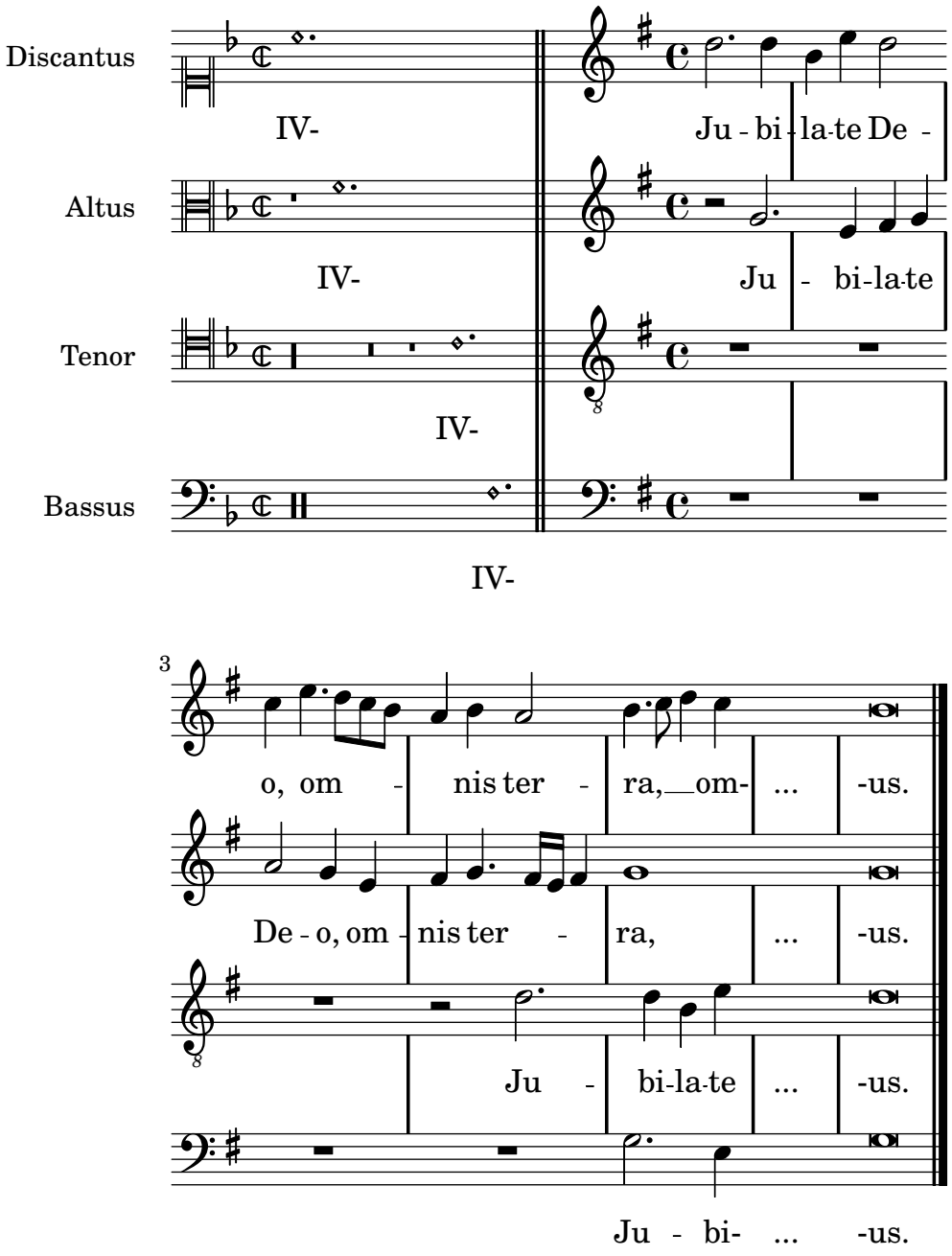
```
  \skip 1*1 % need this extra \skip such that clef change comes
            % after bar line
  \bar ""

  % main
  \revert Score.SpacingSpanner #'spacing-increment % CHECK: no effect?
  \cadenzaOff % turn bar lines on again
  \once \override Staff.Clef #'full-size-change = ##t
  \set Staff.forceClef = ##t
  \key g \major
  \time 4/4
  \override Voice.NoteHead #'style = #'default
  \override Voice.Rest #'style = #'default

  % FIXME: setting printKeyCancellation back to #t must not
  % occur in the first bar after the incipit.  Dto. for forceClef.
  % Therefore, we need an extra \skip.
  \skip 1*1
  \set Staff.printKeyCancellation = ##t
  \set Staff.forceClef = ##f

  \skip 1*7 % the actual music

  % let finis bar go through all staves
  \override Staff.BarLine #'transparent = ##f

  % finis bar
  \bar "|."
}

discantusNotes = {
  \transpose c' c'' {
    \set Staff.instrumentName = "Discantus  "

    % incipit
    \clef "neomensural-c1"
    c'1. s2   % two bars
    \skip 1*8 % eight bars
    \skip 1*1 % one bar

    % main
    \clef "treble"
    d'2. d'4 |
    b e' d'2 |
    c'4 e'4.( d'8 c' b |
    a4) b a2 |
    b4.( c'8 d'4) c'4 |
    \once \override NoteHead #'transparent = ##t c'1 |
    b\breve |
  }
}
```

```
discantusLyrics = \lyricmode {
  % incipit
  IV-

  % main
  Ju -- bi -- |
  la -- te De -- |
  o, om --
  nis ter -- |
  ra, __ om- |
  "..." |
  -us. |
}

altusNotes = {
  \transpose c' c'' {
    \set Staff.instrumentName = "Altus  "

    % incipit
    \clef "neomensural-c3"
    r1        % one bar
    f1. s2    % two bars
    \skip 1*7 % seven bars
    \skip 1*1 % one bar

    % main
    \clef "treble"
    r2 g2. e4 fis g | % two bars
    a2 g4 e |
    fis g4.( fis16 e fis4) |
    g1 |
    \once \override NoteHead #'transparent = ##t g1 |
    g\breve |
  }
}

altusLyrics = \lyricmode {
  % incipit
  IV-

  % main
  Ju -- bi -- la -- te | % two bars
  De -- o, om -- |
  nis ter -- ra, |
  "..." |
  -us. |
}

tenorNotes = {
  \transpose c' c' {
    \set Staff.instrumentName = "Tenor  "
```

```
    % incipit
    \clef "neomensural-c4"
    r\longa   % four bars
    r\breve   % two bars
    r1        % one bar
    c'1. s2   % two bars
    \skip 1*1 % one bar
    \skip 1*1 % one bar

    % main
    \clef "treble_8"
    R1 |
    R1 |
    R1 |
    r2 d'2. d'4 b e' | % two bars
    \once \override NoteHead #'transparent = ##t e'1 |
    d'\breve |
  }
}

tenorLyrics = \lyricmode {
  % incipit
  IV-

  % main
  Ju -- bi -- la -- te | % two bars
  "..." |
  -us. |
}

bassusNotes = {
  \transpose c' c' {
    \set Staff.instrumentName = "Bassus  "

    % incipit
    \clef "bass"
    r\maxima  % eight bars
    f1. s2    % two bars
    \skip 1*1 % one bar

    % main
    \clef "bass"
    R1 |
    R1 |
    R1 |
    R1 |
    g2. e4 |
    \once \override NoteHead #'transparent = ##t e1 |
    g\breve |
  }
}
```

```
bassusLyrics = \lyricmode {
  % incipit
  IV-

  % main
  Ju -- bi- |
  "..." |
  -us. |
}

\score {
  \new StaffGroup = choirStaff <<
    \new Voice =
      "discantusNotes" << \global \discantusNotes >>
    \new Lyrics =
      "discantusLyrics" \lyricsto discantusNotes { \discantusLyrics }
    \new Voice =
      "altusNotes" << \global \altusNotes >>
    \new Lyrics =
      "altusLyrics" \lyricsto altusNotes { \altusLyrics }
    \new Voice =
      "tenorNotes" << \global \tenorNotes >>
    \new Lyrics =
      "tenorLyrics" \lyricsto tenorNotes { \tenorLyrics }
    \new Voice =
      "bassusNotes" << \global \bassusNotes >>
    \new Lyrics =
      "bassusLyrics" \lyricsto bassusNotes { \bassusLyrics }
  >>
  \layout {
    \context {
      \Score

      % no bars in staves
      \override BarLine #'transparent = ##t

      % incipit should not start with a start delimiter
      \remove "System_start_delimiter_engraver"
    }
    \context {
      \Voice

      % no slurs
      \override Slur #'transparent = ##t

      % Comment in the below "\remove" command to allow line
      % breaking also at those barlines where a note overlaps
      % into the next bar.  The command is commented out in this
      % short example score, but especially for large scores, you
      % will typically yield better line breaking and thus improve
      % overall spacing if you comment in the following command.
      %\remove "Forbid_line_break_engraver"
```

```
    }
  }
}
```

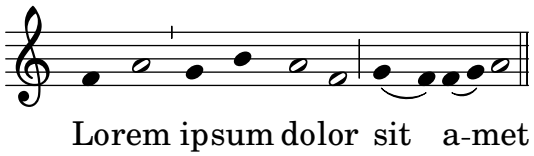


A.5.2 Vorlage zur Transkription von Gregorianik

Dieses Beispiel zeigt eine moderne Transkription des Gregorianischen Chorals. Hier gibt es keine Takte, keine Notenhälse und es werden nur halbe und Viertelnoten verwendet. Zusätzliche Zeichen zeigen die Länge von Pausen an.

```
\include "gregorian-init.ly"
\version "2.11.51"

chant = \relative c' {
  \set Score.timing = ##f
```

```
  f4 a2 \divisioMinima
  g4 b a2 f2 \divisioMaior
  g4( f) f( g) a2 \finalis
}

verba = \lyricmode {
  Lo -- rem ip -- sum do -- lor sit a -- met
}

\score {
  \new Staff <<
    \new Voice = "melody" {
      \chant
    }
    \new Lyrics = "one" \lyricsto melody \verba
  >>

  \layout {
    \context {
      \Staff
      \remove "Time_signature_engraver"
      \remove "Bar_engraver"
      \override Stem #'transparent = ##t
    }
    \context {
      \Voice
      \override Stem #'length = #0
    }
    \context {
      \Score
      barAlways = ##t
    }
  }
}
```



A.6 Jazz-Combo

Hier ist ein ziemlich kompliziertes Beispiel für ein Jazz-Ensemble. Achtung: Alle Instrumente sind in `\key c \major` (C-Dur) notiert. Das bezieht sich auf die klingende Musik: LilyPond transponiert die Tonart automatisch, wenn sich die Noten innerhalb eines `\transpose`-Abschnitts befinden.

```
\version "2.11.51"
\header {
  title = "Song"
  subtitle = "(tune)"
  composer = "Me"
```

```
  meter = "moderato"
  piece = "Swing"
  tagline = \markup {
    \column {
      "LilyPond example file by Amelie Zapf,"
      "Berlin 07/07/2003"
    }
  }
  texidoc = "Jazz tune for combo
             (horns, guitar, piano, bass, drums)."
}

#(set-global-staff-size 16)
\include "english.ly"

%%%%%%%%%%% Some macros %%%%%%%%%%%%%%%%%%

sl = {
  \override NoteHead #'style = #'slash
  \override Stem #'transparent = ##t
}
nsl = {
  \revert NoteHead #'style
  \revert Stem #'transparent
}
cr = \override NoteHead #'style = #'cross
ncr = \revert NoteHead #'style

%% insert chord name style stuff here.

jzchords = { }


%%%%%%%%%%% Keys'n'thangs %%%%%%%%%%%%%%%%

global = {
  \time 4/4
}

Key = { \key c \major }

% ############ Horns ############

% ------ Trumpet ------
trpt = \transpose c d \relative c'' {
  \Key
  c1 c c
}
trpharmony = \transpose c' d {
  \jzchords
}
trumpet = {
```

```
  \global
  \set Staff.instrumentName = #"Trumpet"
  \clef treble
  <<
    \trpt
  >>
}

% ------ Alto Saxophone ------
alto = \transpose c a \relative c' {
  \Key
  c1 c c
}
altoharmony = \transpose c' a {
  \jzchords
}
altosax = {
  \global
  \set Staff.instrumentName = #"Alto Sax"
  \clef treble
  <<
    \alto
  >>
}

% ------ Baritone Saxophone ------
bari = \transpose c a' \relative c {
  \Key
  c1 c \sl d4^"Solo" d d d \nsl
}
bariharmony = \transpose c' a \chordmode {
  \jzchords s1 s d2:maj e:m7
}
barisax = {
  \global
  \set Staff.instrumentName = #"Bari Sax"
  \clef treble
  <<
    \bari
  >>
}

% ------ Trombone ------
tbone = \relative c {
  \Key
  c1 c c
}
tboneharmony = \chordmode {
  \jzchords
}
trombone = {
  \global
```

```
  \set Staff.instrumentName = #"Trombone"
  \clef bass
  <<
    \tbone
  >>
}

% ############ Rhythm Section #############

% ------ Guitar ------
gtr = \relative c'' {
  \Key
  c1 \sl b4 b b b \nsl c1
}
gtrharmony = \chordmode {
  \jzchords
  s1 c2:min7+ d2:maj9
}
guitar = {
  \global
  \set Staff.instrumentName = #"Guitar"
  \clef treble
  <<
    \gtr
  >>
}

%% ------ Piano ------
rhUpper = \relative c'' {
  \voiceOne
  \Key
  c1 c c
}
rhLower = \relative c' {
  \voiceTwo
  \Key
  e1 e e
}

lhUpper = \relative c' {
  \voiceOne
  \Key
  g1 g g
}
lhLower = \relative c {
  \voiceTwo
  \Key
  c1 c c
}

PianoRH = {
  \clef treble
```

```
  \global
  \set Staff.midiInstrument = "acoustic grand"
  <<
    \new Voice = "one" \rhUpper
    \new Voice = "two" \rhLower
  >>
}
PianoLH = {
  \clef bass
  \global
  \set Staff.midiInstrument = "acoustic grand"
  <<
    \new Voice = "one" \lhUpper
    \new Voice = "two" \lhLower
  >>
}

piano = {
  <<
    \set PianoStaff.instrumentName = #"Piano"
    \new Staff = "upper" \PianoRH
    \new Staff = "lower" \PianoLH
  >>
}

% ------ Bass Guitar ------
Bass = \relative c {
  \Key
  c1 c c
}
bass = {
  \global
  \set Staff.instrumentName = #"Bass"
  \clef bass
  <<
    \Bass
  >>
}

% ------ Drums ------
oben = \drummode {
  hh4 <hh sn>4 hh <hh sn> hh <hh sn>4
  hh4 <hh sn>4
  hh4 <hh sn>4
  hh4 <hh sn>4
}

unten = \drummode {
  bd4 s bd s bd s bd s bd s bd s
}

drumContents = {
```

```
  \global
  <<
    \set DrumStaff.instrumentName = #"Drums"
    \new DrumVoice { \voiceOne \oben }
    \new DrumVoice { \voiceTwo \unten }
  >>
}

%%%%%%%% It All Goes Together Here %%%%%%%%%%%%%%%%%%%%

\score {
  <<
    \new StaffGroup = "horns" <<
      \new Staff = "trumpet" \trumpet
      \new Staff = "altosax" \altosax
      \new ChordNames = "barichords" \bariharmony
      \new Staff = "barisax" \barisax
      \new Staff = "trombone" \trombone
    >>

    \new StaffGroup = "rhythm" <<
      \new ChordNames = "chords" \gtrharmony
      \new Staff = "guitar" \guitar
      \new PianoStaff = "piano" \piano
      \new Staff = "bass" \bass
      \new DrumStaff { \drumContents }
    >>
  >>

  \layout {
    \context { \RemoveEmptyStaffContext }
    \context {
      \Score
      \override BarNumber #'padding = #3
      \override RehearsalMark #'padding = #2
      skipBars = ##t
    }
  }

  \midi { }
}
```

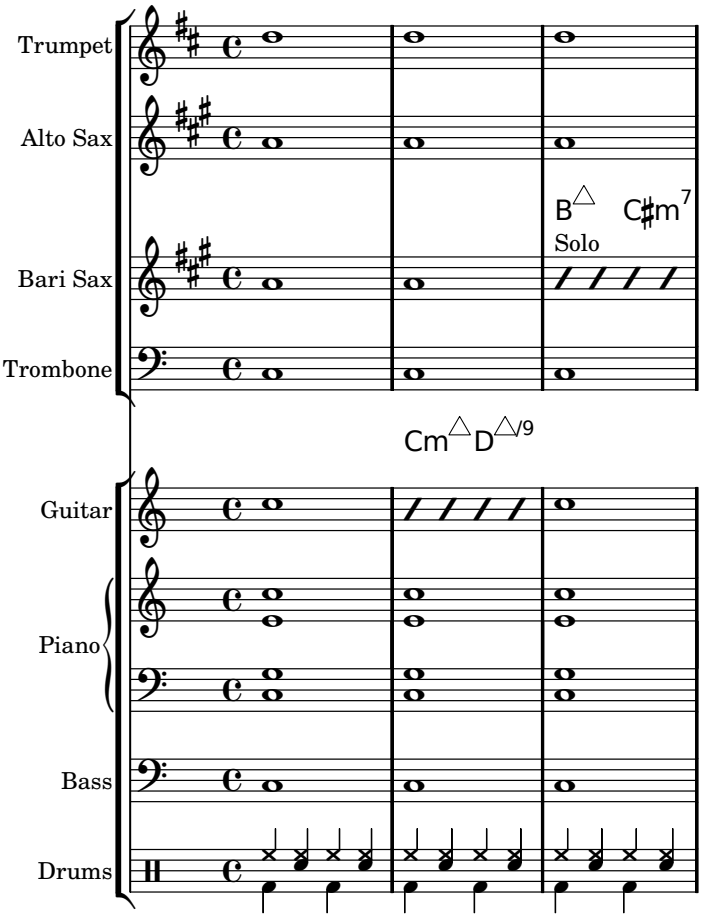
Song

(tune)

Me

moderato

Swing



A.7 Lilypond-book-Vorlagen

Diese Vorlagen können mit `lilypond-book` benutzt werden. Wenn Sie dieses Programm noch nicht kennen, lesen Sie bitte den Abschnitt Abschnitt “LilyPond-book” in *Programmbenutzung*.

A.7.1 LaTeX

LilyPond-Noten können in LaTeX-Dokumente eingefügt werden.

```
\documentclass[]{article}

\begin{document}

Normaler LaTeX-Ttext.

\begin{lilypond}
\relative c'' {
a4 b c d
}
\end{lilypond}

Weiterer LaTeX-Text.

\begin{lilypond}
\relative c'' {
d4 c b a
}
\end{lilypond}
\end{document}
```

A.7.2 Texinfo

LilyPond-Noten können auch in Texinfo eingefügt werden – dieses gesamte Handbuch ist in Texinfo geschrieben.

```
\input texinfo
@node Top

Texinfo-Text

@lilypond[verbatim,fragment,ragged-right]
a4 b c d
@end lilypond

Weiterer Texinfo-Text

@lilypond[verbatim,fragment,ragged-right]
d4 c b a
@end lilypond

@bye
```

Anhang B Scheme-Übung

LilyPond verwendet die Scheme-Programmiersprache sowohl als Teil der Eingabesyntax als auch als internen Mechanismus, um Programmmodule zusammenzufügen. Dieser Abschnitt ist ein sehr kurzer Überblick über die Dateneingabe mit Scheme. Wenn Sie mehr über Scheme wissen wollen, gehen Sie zu http://www.schemers.org.

Das Grundlegendste an einer Sprache sind Daten: Zahlen, Zeichen, Zeichenketten, Listen usw. Hier ist eine Liste der Datentypen, die für LilyPond-Eingabedateien relevant sind.

Boolesche Variablen
: Werte einer Booleschen Variable sind Wahr oder Falsch. Die Scheme-Entsprechung für Wahr ist `#t` und für Falsch `#f`.

Zahlen
: Zahlen werden wie üblich eingegeben, `1` ist die (ganze) Zahl Eins, während `-1.5` ist eine Gleitkommazahl (also eine nicht-ganze).

Zeichenketten
: Zeichenketten werden in doppelte Anführungszeichen gesetzt:

  ```
  "Das ist eine Zeichenkette"
  ```

  Zeichenketten können über mehrere Zeilen reichen:

  ```
  "Das
  ist
  eine Zeichenkette"
  ```

  Anführungszeichen und neue Zeilen können auch mit sogenannten Fluchtsequenzen eingefügt werden. Die Zeichenkette `a sagt "b"` wird wie folgt eingegeben:

  ```
  "a sagt \"b\""
  ```

  Neue Zeilen und Backslashe werden mit `\n` bzw. `\\` eingegeben.

In einer Notationsdatei werden kleine Scheme-Abschnitte mit der Raute (`#`) eingeleitet. Die vorigen Beispiele heißen also in LilyPond:

```
##t ##f
#1 #-1.5
#"Das ist eine Zeichenkette"
#"Das
ist
eine Zeichenkette"
```

Für den Rest dieses Abschnitts nehmen wir an, dass die Daten immer in einer LilyPond-Datei stehen, darum wird immer die Raute verwendet.

Scheme kann verwendet werden, um Berechnungen durchzuführen. Es verwendet eine *Präfix*-Syntax. Um 1 und 2 zu addieren, muss man `(+ 1 2)` schreiben, und nicht $1 + 2$, wie in traditioneller Mathematik.

```
#(+ 1 2)
  ⇒ #3
```

Der Pfeil ⇒ zeigt an, dass das Ergebnis der Auswertung von `(+ 1 2)` `3` ist. Berechnungen können geschachtelt werden und das Ergebnis einer Berechnung kann für eine neue Berechnung eingesetzt werden.

```
#(+ 1 (* 3 4))
  ⇒ #(+ 1 12)
  ⇒ #13
```

Diese Berechnungen sind Beispiele von Auswertungen. Ein Ausdruck wie `(* 3 4)` wird durch seinen Wert `12` ersetzt. Ähnlich verhält es sich mit Variablen. Nachdem eine Variable definiert ist:

```
twelve = #12
```

kann man sie in Ausdrücken weiterverwenden:

```
twentyFour = #(* 2 twelve)
```

Die 24 wird in der Variablen `twentyFour` gespeichert. Die gleiche Zuweisung kann auch vollständig in Scheme geschrieben werden:

```
#(define twentyFour (* 2 twelve))
```

Der *Name* einer Variable ist auch ein Ausdruck, genauso wie eine Zahl oder eine Zeichenkette. Er wird wie folgt eingegeben:

```
#'twentyFour
```

Das Apostroph `'` verhindert, dass bei der Scheme-Auswertung `twentyFour` durch `24` ersetzt wird. Anstatt dessen erhalten wir die Bezeichnung `twentyFour`.

Diese Syntax wird sehr oft verwendet, weil es manche Einstellungsveränderungen erfordern, dass Scheme-Werte einer internen Variable zugewiesen werden, wie etwa

```
\override Stem #'thickness = #2.6
```

Diese Anweisung verändert die Erscheinung der Notenhälse. Der Wert `2.6` wird der Variable `thickness` (Dicke) eines `Stem`-(Hals)-Objektes gleichgesetzt. `thickness` wird relativ zu den Notenlinien errechnet, in diesem Fall sind die Hälse also 2,6 mal so dick wie die Notenlinien. Dadurch werden Hälse fast zweimal so dick dargestellt, wie sie normalerweise sind. Um zwischen Variablen zu unterscheiden, die in den Quelldateien direkt definiert werden (wie `twentyFour` weiter oben), und zwischen denen, die für interne Objekte zuständig sind, werden hier die ersteren „Bezeichner“ genannt, die letzteren dagegen „Eigenschaften“. Das Hals-Objekt hat also eine `thickness`-Eigenschaft, während `twentyFour` ein Bezeichner ist.

Sowohl zweidimensionale Abstände (X- und Y-Koordinaten) als auch Größen von Objekten (Intervalle mit linker und rechter Begrenzung) werden als `pairs` (Paare) eingegeben. Ein Paar[1] wird als `(erster . zweiter)` eingegeben und sie müssen mit dem Apostroph eingeleitet werden, genauso wie Symbole:

```
\override TextScript #'extra-offset = #'(1 . 2)
```

Hierdurch wird das Paar (1, 2) mit der Eigenschaft `extra-offset` des TextScript-Objektes verknüpft. Diese Zahlen werden in Systembreiten gemessen, so dass der Befehl das Objekt eine Systembreite nach rechts verschiebt und zwei Breiten nach oben.

Die zwei Elemente eines Paares können von beliebigem Inhalt sein, etwa

```
#'(1 . 2)
#'(#t . #f)
#'("blah-blah" . 3.14159265)
```

Eine Liste wird eingegeben, indem die Elemente der Liste in Klammern geschrieben werden, mit einem Apostroph davor. Beispielsweise:

```
#'(1 2 3)
#'(1 2 "string" #f)
```

Die ganze Zeit wurde hier schon Listen benutzt. Eine Berechnung, wie `(+ 1 2)`, ist auch eine Liste (welche das Symbol `+` und die Nummern 1 und 2 enthält. Normalerweise werden Listen als Berechnungen interpretiert und der Scheme-Interpreter ersetzt die Liste mit dem Ergebnis der Berechnung. Um eine Liste an sich einzugeben, muss die Auswertung angehalten werden. Das geschieht, indem der Liste ein Apostroph vorangestellt wird. Für Berechnungen kann man also den Apostroph nicht verwenden.

Innerhalb einer zitierten Liste (also mit Apostroph) muss man keine Anführungszeichen mehr setzen. Im Folgenden ein Symbolpaar, eine Symbolliste und eine Liste von Listen:

[1] In der Scheme-Terminologie wird ein Paar `cons` genannt und seine zwei Elemente `car` und `cdr`.

```
#'(stem . head)
#'(staff clef key-signature)
#'((1) (2))
```

Anhang C GNU Free Documentation License

Version 1.1, March 2000

0. PREAMBLE

   The purpose of this License is to make a manual, textbook, or other written document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

   This License is a kind of ‚copyleft‘, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

   We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

   This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The ‚Document‘, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as ‚you‘.

   A ‚Modified Version‘ of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

   A ‚Secondary Section‘ is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

   The ‚Invariant Sections‘ are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

   The ‚Cover Texts‘ are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

   A ‚Transparent‘ copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file

format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not ‚Transparent' is called ‚Opaque'.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The ‚Title Page' means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, ‚Title Page' means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

   You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

   You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

   It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section entitled ‚History', and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled ‚History' in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the ‚History' section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. In any section entitled ‚Acknowledgments' or ‚Dedications', preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgments and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section entitled ‚Endorsements'. Such a section may not be included in the Modified Version.

N. Do not retitle any existing section as ‚Endorsements' or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to

the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled ‚Endorsements', provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled ‚History' in the various original documents, forming one section entitled ‚History'; likewise combine any sections entitled ‚Acknowledgments', and any sections entitled ‚Dedications'. You must delete all sections entitled ‚Endorsements.'

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an ‚aggregate', and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License ‚or any later version‘ applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

Anhang: Wie kann die Lizenz für eigene Dokumente verwendet werden

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
  Copyright (C)  year  your name.
  Permission is granted to copy, distribute and/or modify this document
  under the terms of the GNU Free Documentation License, Version 1.1
  or any later version published by the Free Software Foundation;
  with the Invariant Sections being list their titles, with the
  Front-Cover Texts being list, and with the Back-Cover Texts being list.
  A copy of the license is included in the section entitled ,GNU
Free Documentation License‘
.
```

If you have no Invariant Sections, write ‚with no Invariant Sections‘ instead of saying which ones are invariant. If you have no Front-Cover Texts, write ‚no Front-Cover Texts‘ instead of ‚Front-Cover Texts being *list*‘; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Anhang D LilyPond-Index

#

<

\

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

Z