Katholieke
Universiteit
Leuven

Faculteiten Wetenschappen
en Toegepaste Wetenschappen

COMPONEREN VAN POLYFONE MUZIEK

Bart VRANCKEN

Eindverhandeling aangeboden tot het
behalen van de graad van licentiaat
in de informatica

2002–2003

Promotor: Prof. Dr. D. DE SCHREYE

Faculteiten Wetenschappen
en Toegepaste Wetenschappen

K.U. Leuven
Academiejaar 2002–2003

Naam en voornaam student: Vrancken Bart

Titel:

Componeren van polyfone muziek

Engelse vertaling:

Composing polyphone music

ACM Classificatie: J.5 Arts and Humanities—Performing arts (e.g., dance, music), H.5.5 Sound and Music Computing—Methodologies and techniques , D.1.6 Logic Programming, D.3.3 Language Constructs and Features—Constraints

Korte inhoud:

Om een gegeven melodie vierstemmig te maken, bestaan er verschillende regels, die afhankelijk zijn per tijdperk. In de klassieke muziekleer zijn de 2 belangrijkste vormen harmonie en contrapunt.

Constraint Logic Programming is een paradigma dat gesteund is op het modelleren van een probleem aan de hand van regels. De gebruiker definieert eerst het domein waar de veranderlijken zich situeren en daarna de regels waaraan ze zich moeten houden. Vervolgens geeft het programma aan of dat het probleem op te lossen is en geeft eventueel een oplossing.

Met deze twee basisgegevens werken we in de tekst een programma uit om koormuziek te genereren uit een gegeven melodie volgens de regels van harmonie en contrapunt.

Eindverhandeling aangeboden tot het behalen
van de graad van licentiaat in de informatica

Promotor: Prof. Dr. D. De Schreye

Assessoren: A. Serebrenik
P. Bekaert

Begeleiders: A. Serebrenik
D. Gilis

Dankwoord

Hierbij zou ik enkele mensen willen bedanken, die het mogelijk hebben gemaakt om dit eindwerk te maken.

In de eerste plaats mijn promotor Danny De Schreye, waar ik altijd welkom was en die met veel interesse luisterde.

Mijn twee begeleiders Alexander Serebrenik en David Gilis. Zonder hun steun, hun opbouwende kritiek en hun onvoorstelbare inzet zou dit werk er heel anders uitgezien hebben.

Mijn ouders, die mij aangemoedigd hebben om deze studies aan te vatten en deze ook hebben gesponsord.

Tenslotte wil ik Kristien bedanken. Zonder haar liefdevolle steun was ik er nooit toegekomen om deze thesis tot een goed einde te brengen.

Inhoudsopgave

Lijst van tabellen

Lijst van figuren

Listings

Hoofdstuk 1

Inleiding

1.1 Probleemstelling

We bevinden ons in het domein van de muziek. Meer bepaald is er voor gekozen om de problematiek van koormuziek aan te pakken. In de meeste koren zijn er minstens vier stemmen aanwezig, namelijk twee van de vrouwen, de sopraan en de alt, en twee van de mannen, de tenor en de bas. Het gebeurt echter dikwijls dat een dirigent een partituur krijgt met slechts een éénstemmige melodie. Soms wordt er dan voor gekozen, dat alle stemmen die zelfde melodie zingen, zodat er geen verschil tussen de stemmen te horen is. Meestal wil men echter gebruik maken van het verschil en de rijkheid aan stemmen en kiest men ervoor om de gegeven melodie vierstemmig te maken.

Behalve de vraag op muzikaal gebied over de manier waarop men een muziekstuk vierstemmig moet maken, moet men zich ook afvragen hoe men dit een computer zal aanleren. Doordat men met veel parameters moet rekening houden, moet men een efficiënte manier gebruiken, opdat de oplossing in een beperkte tijd gevonden wordt. Deze veelheid van regels levert ook niet een enige oplossing op, dus moet er ook nagekeken worden, wat uiteindelijk als beste oplossing mag weerhouden worden.

1.2 Oplossingen

Zomaar willekeurig wat noten schrijven onder die melodie zou geen mooie stemmen opleveren, die mooi samen klinken en afzonderlijk ook mooi te zingen zijn. In het verleden zijn er verschillende regels geweest, waaraan een melodie moet voldoen om mooi geacht te worden. Net zoals in elke kunstvorm varieert het begrip van schoonheid door de verschillende tijden en is elk nieuw tijdperk een antwoord op de voorgaande eeuwen. Een kunststroming werd meestal niet bepaald door regels, maar de regels werden bepaald doordat contemporaine artiesten een zelfde stijl

hanteerden, die brak met de vorige stroming.

In de klassieke muziekleer worden de regels van *harmonie* en *contrapunt* bekeken. Historisch gezien zouden we moeten beginnen met de leer van het contrapunt. Didactisch gezien wordt er meestal eerst de harmonie aangeleerd, omdat die toegankelijker is als muziekleer.

1.2.1 Homofonie: Harmonie

De harmonieleer wordt soms ook *homofonie* genoemd. Met homofonie bedoelt men dat er één stem de belangrijkste is en dat de andere stemmen deze stem begeleiden zodat ze beter tot haar recht komt. Men bekijkt telkens per noot welke noten er in de andere stemmen moeten klinken. Dit is een verticale zienswijze van het componeren.

1.2.2 Polyfonie: Contrapunt

Als we dit anders aanpakken en horizontaal gaan kijken, dan houden we ons bezig met de melodie en komt de *melodieleer* ter sprake. Deze melodieleer wordt contrapunt genoemd en bestaat er in om via de gegeven melodie een andere melodie op te bouwen. Daarna wordt er een derde stem toegevoegd en dan de vierde. De nadruk ligt hier op het vormen van mooie melodiëen. Hierdoor is elke melodie even waardig en spreekt men van *polyfonie*, meerstemmigheid.

1.3 Implementatie

Voor de implementatie hadden we een waaier aan mogelijkheden aan programmeertalen. We kozen voor dit probleem *constraint logic programming* (CLP). Dit is een logische keuze omwille van de declaratieve kracht van deze vorm van programmeren die belangrijk is bij een goede kennisrepresentatie en omdat dit concept werkt door middel van regels. Zoals hiervoor reeds gezegd, bestaan er van de verschillende stijlen van de muziek regels en kan dit probleem gezien worden als een planningsprobleem, waar CLP uiterst geschikt voor is. Kort gezegd zoekt CLP voor de variabelen, die zich in een gegeven domein bevinden, oplossingen die aan de gegeven regels voldoen. Deze regels beperken het aantal oplossingen door het domein van de variabelen te verkleinen. In de eerste plaats zal CLP zeggen of er oplossingen zijn, daarna zal het de oplossingen geven, indien dit gewenst is.

Deze problematiek zou ook in een programmeertaal zoals Java of C++ kunnen geïmplementeerd worden, maar dan zouden we de concepten in de muziek als objecten moeten voorstellen. Het nadeel hiervan is, dat je niet meer declaratief te werk kunt gaan, en dat je de interactie moet gaan modelleren tussen de objecten aan de

hand van de regels, terwijl we via CLP een opsomming van regels kunnen geven aan de probleemoplosser, die dit pakket dan verwerkt en een oplossing geeft.

1.4 Doel van dit eindwerk

Het is de bedoeling om een programma af te leveren dat zowel via de harmonie als via het contrapunt een oplossing kan bieden voor dirigenten, die geen tijd hebben om een muziekstuk vierstemmig te maken. Hierbij wordt er ook een vergelijking gemaakt, zodat we kunnen zien of deze twee methodes tot dezelfde oplossing komen.

Het is ook de bedoeling om de kracht van het paradigma van CLP te tonen om dit soort problemen op te lossen, die te complex zijn om gewoon alle oplossingen te overlopen en te testen om tot een goede oplossing te komen.

1.5 Overzicht van de tekst

De tekst handelt over de muziektheorie en over het gebruik van CLP. In het tweede hoofdstuk wordt er een inleiding gegeven over algemene muzikale begrippen. Het derde hoofdstuk doet hetzelfde voor CLP. In het vierde hoofdstuk wordt de overgang gemaakt tussen deze twee en wordt er weergegeven hoe we de muziek zullen voorstellen in het programma. Dan volgen 2 hoofdstukken over de specifieke stijlen van de homofonie en de polyfonie samen met hun implementatie. Tenslotte sluiten we af met verschillende resultaten.

Samengevat ziet de tekst er als volgt uit:

1. Inleiding:
    - Muziektheorie
    - CLP
2. Kennisrepresentatie
3. Implementatie:
    - Homofonie
    - Polyfonie
4. Resultaten

Hoofdstuk 2

Inleiding tot de algemene muziektheorie

In dit hoofdstuk wordt er een eenvoudige inleiding gegeven over de belangrijkste begrippen in de muziektheorie, die later zowel in de harmonie als in het contrapunt zullen terug komen.

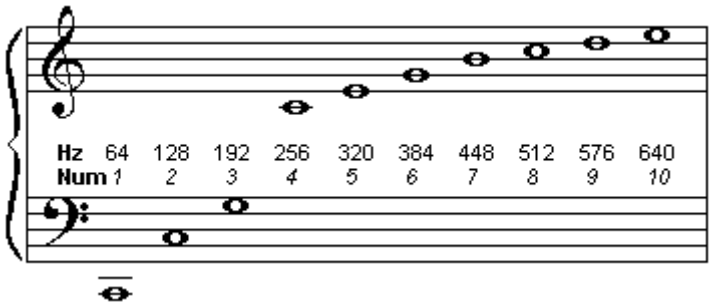
2.1 Over tonen en intervallen

Een *toon* is een regelmatige luchttrilling met een frequentie hoger dan ca. 20 Hz en lager dan ca. 15.000 Hz. Een normaal menselijk oor kan onder normale omstandigheden ongeveer 850 verschillende toonhoogten waarnemen. Onder zeer gunstige omstandigheden kan het geoefende oor in dit zelfde gebied ca. 1500 tonen waarnemen.

In de muziek worden deze mogelijke waarneembare toonhoogten niet allemaal gebruikt. Men heeft een keuze gemaakt, die men als aangenaam gewaar wordt in een bepaalde cultuur. Culturen veranderen met de jaren en ook kan die keuze dan soms aangepast worden. De keuze van verschillende tonen tussen 20 Hz en 15.000 Hz noemt men het *toonsysteem*. Merkwaardig genoeg zijn de bekendste toonsystemen gebaseerd op de notie van het octaaf en zijn onderverdeling.

Een *octaaf* wordt gevormd door twee tonen waarvan de frequentie van de ene toon het dubbele is van de frequentie van de andere toon. Dit wil zeggen dat een toon met frequentie 220 Hz, een octaaf vormt met een toon met frequentie 440 Hz. Deze laatste frequentie is de bekendste frequentie in onze cultuur, omdat die gebruikt wordt in een “normale” stemvork voor de la.

Om een octaaf te herkennen moet men deze niet gaan opmeten. Ons gehoor vindt die afstand zo natuurlijk, dat de twee tonen enorm op elkaar gelijken. Enkel twee tonen van een gelijke frequentie klinken nog meer hetzelfde, ook al is het onderscheid met een octaaf soms moeilijk te maken. Dit wordt nog versterkt dat de



Figuur 2.1: Boventonen

octaaf ook is opgenomen in de reeks van *boventonen* van een toon. Boventonen zijn die tonen die meetrillen met de gespeelde toon. De frequentie van deze boventonen zijn een veelvoud van de frequentie van de aangespeelde toon. In figuur 2.1 zie je de boventonen van een bepaalde toon. (Voor de notatie, zie paragraaf 2.4)

Het octaaf zelf wordt dan nog op een bepaalde manier ingedeeld en de tonen die men op die manier krijgt, verhouden zich naar elkaar toe in verschillende *intervallen*. Zo is een octaaf ook een interval. Verschillende theoretici, zoals Pythagoras, Zarlino, Euler en anderen hebben die intervallen verklaard als verhoudingen van frequenties. Hier dieper op ingaan zou ons te ver leiden.

De indeling van het octaaf zou op verschillende manieren kunnen gebeuren. De meest gebruikelijke is om die in te delen in afstanden, die aangenaam zijn om te horen. Dit bekomt men door het octaaf in twaalf stukken te delen. De lengte van elk stuk (de afstand van één van zijn kanten tot de andere) noemt men *een halve toon*. Twee halve tonen samen vormen een hele toon.

Op een piano kan je een bepaald motief zien in de zwarte en witte toetsen. Zo is er een afwisseling van groepjes met twee zwarte en groepjes met drie zwarte toetsen. Als je een volledig motiefje neemt, tel je twaalf toetsen en heb je een octaaf gevonden. De andere intervallen zijn ook gemakkelijk af te leiden op een piano. Het interval van de *priem* is, als je een toon aanslaat en dezelfde toon opnieuw laat klinken. Dit is dus als er geen afstand is tussen de twee toetsen. (Het concept van intervallen en een gelijkaardige naamgeving stamt reeds af van de klassieke oudheid, terwijl het getal nul pas rond 1200 na Christus in Europa bekend wordt.) Het verschil van de *secunde* krijg je als je twee witte toetsen langs elkaar laat klinken. Bij een terts zit er een witte toets tussen de twee andere. Zo gaat het verder met de *kwart, kwint, sext, septiem* tot aan het octaaf, waar er zes witte toetsen tussen liggen.

Als we de intervallen bekijken, telkens vertrekkende van een do (zie verder voor de definitie van een do) en we nemen telkens de afstand tussen die do en een hogere witte toets tot aan de volgende do dan hebben we de *reine* en de *grote* intervallen. Als je de afstand van die intervallen neemt en van de do naar beneden gaat over die afstand dan kom je soms terug op een witte toets uit. Dit noemen we de reine

intervallen en dit is het geval bij een kwart, kwint of octaaf. Doen we hetzelfde bij de andere intervallen (de grote) dan komen we uit op een zwarte toets. Een halve toon lager dan een groot interval is het gelijknamige *kleine* interval. Elk interval kan men nog eens een halve toon verlagen (*verminderd interval*) of verhogen (*overmatig interval*). Samengevat kunnen priemen, kwarten en kwinten en octaven rein zijn en niet groot of klein en kunnen de andere intervallen alles behalve rein zijn. Alle intervallen kunnen verminderd of overmatig zijn. Vooraf gegaan door hun toonafstand, zijn in tabel 2.1 de dertien "normale" intervallen samengevat. Grotere intervallen dan een octaaf bestaan ook als samenstelling van een octaaf met een andere afstand.

2.2 Over tonen en namen

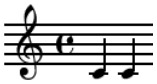
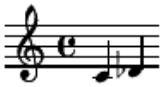
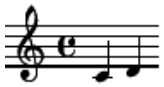
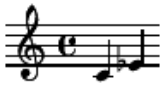
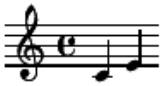
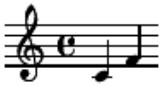
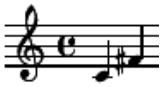
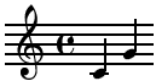
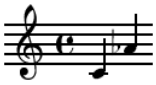
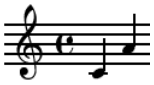
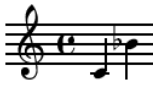
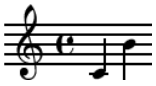
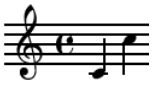
Zoals hiervoor reeds besproken, is de frequentie van 440 Hz de frequentie van een gewone la. Deze naamgeving wordt toegeschreven aan Guido d'Arezzo, een bekende musicoloog uit de tiende eeuw na Christus. Hij was de eerste die series van zes tonen gebruikte om de muziek te benoemen in tegenstelling tot de oude Grieken die er slechts vier gebruikten. Zijn naamgeving was afkomstig van een bekend muziekstuk uit zijn tijd, de heilige Johannes Hymne (achtste eeuw), waar hij telkens de eerste lettergreep van elk vers als naam voor de toon bombardeerde.

1. **Ut** queant laxis
2. **Re**sonare fibris
3. **Mi**ra gestorum
4. **Fa**muli tuorum
5. **Sol**ve poluti
6. **La**bii reatum,

   Sancte Ioannes!

De Ut werd later hernoemd tot *do*. Toen de zevende toon veel gebruikt werd, heeft men die genoemd naar de de initialen van het laatste vers, **S**ancte **I**oannes. Voor deze namen aan te leren zijn er in elke taal wel gedichtjes of liedjes gekomen, die spelenderwijs de notennamen aanleerde. Hieronder een voorbeeld uit de musical Sound of Music:

> do, dat is een wiegelied
> re, een hert dat je niet ziet
> mi, dat zegt men voor Marie
> fa, begin van fantasie

Grootte	*Middeleeuwse naam*	*Hedendaagse naam*	*Voorbeeld*
geen afstand	unisonus	reine prime	
halve toon	semitonium	kleine secunde	
hele toon	tonus	grote secunde	
anderhalve toon	semiditonus	kleine terts	
twee tonen	ditonus	grote terts	
twee-en-een-halve toon	diatessaron	reine kwart	
drie tonen	tritonus	overmatige kwart	
drie-en-een-halve toon	diapente	reine kwint	
vier tonen	diapente cum semitonio	kleine sext	
vier-en-een-halve toon	diapente cum tono	grote sext	
vijf tonen	diapente cum semiditono	kleine septiem	
vijf-en-een-halve toon	diapente cum ditono	grote septiem	
zes tonen	diapason	rein octaaf	

Tabel 2.1: De intervallen

sol, dat is de spaanse zon,
la, een lied dat pas begon
si, begin van symfonie
en dat brengt ons weer bij do

do mi mi, mi sol sol
re fa fa, la si si

sol do la fa mi do re
sol do la si do re do

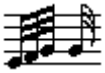
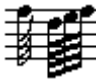
Buiten de naamgeving van do, re, mi, fa, sol, la, si is er in de muziek ook een letternotatie, die reeds door de Grieken en Romeinen werd gebruikt en die voor het eerst gedocumenteerd is door de Romeinse schrijver Boethius[1], die vijf eeuwen voor Guido d'Arezzo leefde. In die notatie wordt de la geschreven als een *a*, een si als een *b*, en zo verder tot de sol, die wordt geschreven als een *g*. Om te weten in welk octaaf de noot gelegen is, kan men er nog het octaafnummer bijzetten, zodat de la op frequentie 440 Hz, *a4* genoemd wordt, die op 220 Hz, *a3* en zo verder.

Op de muziekpartituur staan ook nog de tekens van een kruis (♯) en die van een mol (♭), die de toon respectievelijk een halve toon verhogen of verlagen. Dit wordt vertaald in lettercode naar *is* en naar *es*. Zo is een fa kruis gelijk aan *fis* en een si mol gelijk aan *bes*. Met klinkers valt de e van *es* weg, zodat *as* gelijk is aan la mol en *es* aan mi mol. Sommige tonen vallen samen op een piano zoals *as* en *gis* op één zwarte toets. Deze tonen hebben echter een verschillende functie in een melodie, die op andere instrumenten zoals een viool zelfs een zeer kleine andere toonhoogte hebben. (Deze afstand heeft de grootte van één accent. Dit is de kleinst waarneembare afstand voor het menselijke gehoor.)

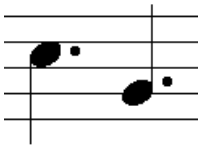
2.3 Van tonen naar noten

Over het algemeen worden de begrippen tonen en noten door elkaar gebruikt. De benaming van noten slaagt echter terug op het teken dat men gebruikt om een toon aan te duiden. Als men naar de tekening van een *hele noot* kijkt, dan zie je de gelijkenis met de gelijknamige boomvrucht. Een hele noot is een noot die een eenheid lang duurt. Deze eenheid is niet absoluut en verschilt meestal van stuk tot stuk en van uitvoering tot uitvoering. De noten met een andere lengte verhouden zich echter tot deze noot. In tabel 2.2 staan de verschillende soorten noten opgesomd, voorafgegaan door hun verhouding met een hele noot. De naamgeving wordt hier ook volledig op gebaseerd. (een *halve noot*, een *vierde noot*, een *achtste noot*, . . .)

Tenslotte kan men de nootwaarde nog met een half verlengen door er een punt achter te zetten. Zo duurt een vierde noot met een puntje erachter zoals in figuur 2.2, even lang als een vierde noot gevolgd door een achtste noot.

1	Een hele noot: 
$\frac{1}{2}$	Een halve noot: 
$\frac{1}{4}$	Een vierde noot: 
$\frac{1}{8}$	Een achtste noot: 
$\frac{1}{16}$	Een zestiende noot: 
$\frac{1}{32}$	Een tweendertigste noot: 
$\frac{1}{64}$	Een vierenzestigste noot: 

Tabel 2.2: De verschillende soorten noten

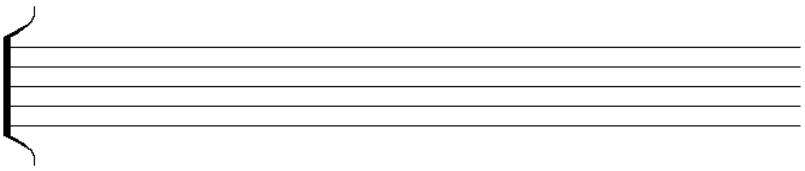


Figuur 2.2: Gepunte noten

Een muziekstuk mag natuurlijk niet overladen wordt met klank. Het zou trouwens slachtoffers maken bij zangers en blazers als zij niet af en toe de ruimte krijgen om te ademen. We gebruiken hiervoor *rusten* van een bepaalde duur. Net als noten worden rusten afgeleid van een hele rust zoals je kan aflezen in tabel 2.3. Ook hier kan een punt de rust voor de helft verlengen.

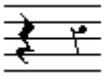
2.4 Over notenbalken en sleutels

Vanaf de zevende eeuw na Christus was er sprake van een grafische notatie voor de muziek. Deze notatie gaf bij benadering de verschillende toonhoogtes weer en was bedoeld als geheugensteuntje of als spiekbriefje voor de minstrelen en de monniken die de liederen reeds kenden. Er was toen nog geen sprake van de huidige notenbalken en sleutels en daardoor werd de muziek onnauwkeurig beschreven. Vanaf de veertiende eeuw is ons hedendaagse notensysteem ontwikkeld in Europa en heeft het zich verspreid over de hele wereld, zodat deze taal, de meest herkenbare internationale taal is, die er ooit bestaan heeft. Om tot het huidige systeem te komen, heeft men verschillende methodes en tekens gebruikt, waarvan enkel die overbleven, die nuttig waren voor de toenmalige muziek. De rest is bijna allemaal vergeten, wat het moeilijk maakt om sommige oudere muziekstukken te ontcijferen die gebruik maken van tekens, die nu niet meer verstaan worden.



Figuur 2.3: De notenbalk

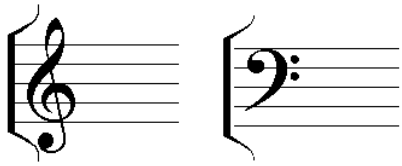
Een moderne *notenbalk* (figuur 2.3) bestaat uit 5 horizontale evenwijdige lijntjes, die twee aspecten van de muziek vertolken. De tijd wordt horizontaal voorgesteld, beginnende van links naar rechts. De afstand is echter niet consistent of scaleerbaar, zodat je twee muziekstukken niet in tijd kunt vergelijken door naar de notenbalken te kijken. Vertikaal wordt de toonhoogte voorgesteld. Een toon met

1	Een hele rust: 
$\frac{1}{2}$	Een halve rust: 
$\frac{1}{4}$	Een vierde rust: 
$\frac{1}{8}$	Een achtste rust: 
$\frac{1}{16}$	Een zestiende rust: 
$\frac{1}{32}$	Een tweendertigste rust: 
$\frac{1}{64}$	Een vierenzestigste rust: 

Tabel 2.3: De verschillende soorten rusten

een hogere frequentie wordt hoger geplaatst op de notenbalk dan een toon met een lagere frequentie. Deze frequentie is net zoals de tijd niet absoluut bepaald door de notenbalk. Het eerste teken dat op de notenbalk staat, zorgt ervoor dat de frequentie wel absoluut bepaald wordt. Dat teken is als het ware de *sleutel* tot het lezen van die notenbalk en wordt dan ook zo genoemd.

Er zijn verschillende sleutels, waarvan de Sol-sleutel en de Fa-sleutel de bekendste zijn. De eerste sleutel op de notenbalk van figuur 2.4 is de Sol-sleutel en is herkenbaar als een oude letter G. Je kan de sol vinden op de tweede lijn van de notenbalk, waar de sleutel met zijn krul begint. De tweede sleutel is een Fa-sleutel, herkenbaar door de oude letter F, gevolgd door twee puntjes, die aanduiden waar je de fa kunt vinden op de notenbalk. De gevonden Fa en Sol bevinden zich respectievelijk in het midden van het bereik van de mannen (midden van laagste bas-noot en hoogste tenor-noot) en vrouwenstemmen (midden van laagste alt-noot en hoogste sopraan-noot).

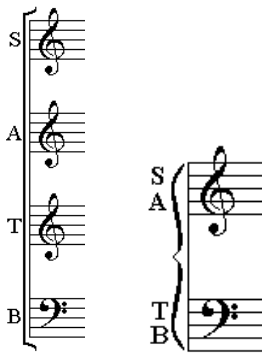


Figuur 2.4: De sol, de fa en de tenorsleutel

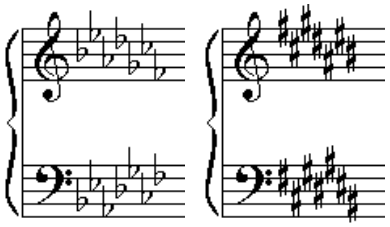
Figuur 2.5 vat dit dan samen om koormuziek weer te geven. De letters S,A,T,B geven aan, waar men de sopraan, alt, tenor en bas kan vinden. De *sopraan* is de hoogste vrouwenstem, die een bereik heeft van een lage do (c4) tot de hoge la (a5). De *alt* is de lage vrouwenstem, met een bereik van de lage fa (f3) tot de hoge re (d5). De *tenor* en de *bas* liggen respectievelijk een octaaf lager dan de sopraan en de alt[2]. Links wordt de lange notatie gegeven, waar men voor de tenoren een solsleutel gebruikt. Rechts wordt een afgekorte notatie weergegeven, waar de vrouwenstemmen op de solsleutel worden geschreven en de mannenstemmen op de fasleutel.

Behalve de sleutel moet elke notenbalk beginnen met de wijzigingstekens (♭ of ♯) die bij de toonladder (zie verder) horen. Dit wordt de *voortekening* genoemd en is een deelverzameling van figuur 2.6, waar er wat kruizen of mollen zijn verwijderd langs de rechterkant. Deze wijzigingstekens zorgen ervoor dat de toonhoogte van die noten op de notenbalk verhoogd of verlaagd worden. In een melodie kan de wijziging ongedaan worden door een herstellingsteken (♮) of een ander wijzigingsteken.

Maten zijn een onderverdeling van het gehele muziekstuk. Ze geven aanwijzingen aan de muzikant omtrent nadrukken, die in de muziek gelegd moeten worden. Zo accentueert men de eerste noot na een maatstreep meestal, zodat die duidelijker overkomt. Een maat wordt aangeduid door een verticale streep die begint aan het onderste lijntje van de notenbalk en eindigt aan het bovenste. Het einde van een

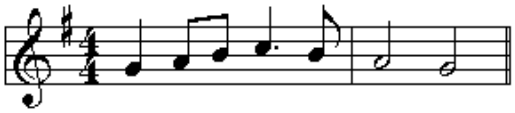


Figuur 2.5: Sleutels voor koormuziek



Figuur 2.6: De voortekening

stuk wordt aangeduid door een dubbele maatstreep. In het begin van elk muziekstuk staat er een tijdsaanduiding naar de lengte van een maat (dit is de *maatsoort*). Het onderste cijfer van de tijdsaanduiding is de referentielengte van de noten en bovenaan staat er hoeveel noten met die lengte erin kunnen zitten. De aanduiding in figuur 2.7 wil zeggen dat elke maat een lengte heeft van vier keer de lengte van een vierde noot.



Figuur 2.7: Een voorbeeldmelodie

2.5 Toonladders

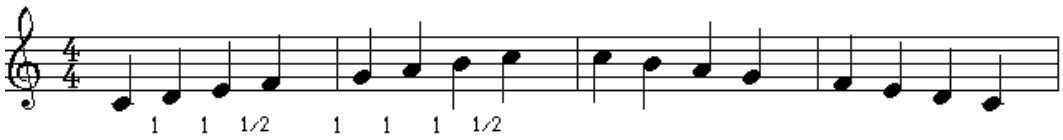
Een *toonladder* is een trapsgewijze volgorde van tonen, waarvan het aantal en de afstand in verschillende tijdperken en culturen verschillend zijn. De vorming van de toonladder is afhankelijk van het aantal in een *toonsysteem* ter beschikking staande tonen, alsmede van de uit

> de vocale tradities of de structuur van de instrumenten voortkomende intervalafstanden tussen de afzonderlijke tonen (zie ook stemming). In het algemeen worden toonladders beschouwd als het basismateriaal van de verschillende muzikale systemen. [3]

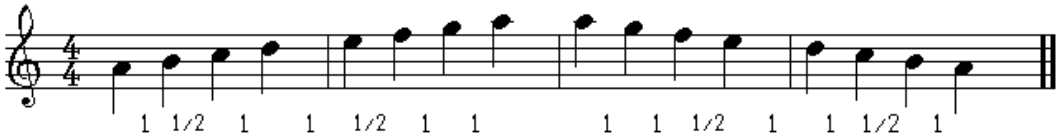
Het meest gebruikelijke systeem in onze cultuur is de diatonische[1] toonladder. In deze toonladder ontbreken er enkele van de twaalf halve tonen. Men onderscheidt er de verschillende intervallen, priem, secunde tot het octaaf en dus in totaal acht noten van de dertien (twaalf halve tonen plus de octaaftoon).

Een *toonaard* is de aard van de toonladder, die de melodie in een andere stemming kan brengen, droeviger maken of vreugdevoller. Hier wordt het onderscheid gemaakt tussen een *grote* of *kleine* toonladder. Een grote toonaard klinkt vreugdevoller dan een kleine toonaard, omdat het interval van de terts een grote terts is, terwijl die van de kleine toonaard een kleine terts is. De kleine toonladder die zich een kleine terts lager bevindt dan een grote toonladder en die grote toonladder, noemt men de *parallelle toonladders* van elkaar. In de praktijk zijn er nog meerdere, minder bekende toonaarden en zelfs enkele toonladders specifiek voor een bepaalde muziekstijl (denk bijvoorbeeld aan de *bluestoonladders* of aan de *kerktoonladders*).

Om het standaardvoorbeeld te geven bekijken we de toonladders die enkel witte toetsen hebben, die van do groot en la klein. Op de tekeningen staan ook de verschillende toonafstanden bij[4]. Op tekening 2.8 van do groot zie je dat er tussen de do en de mi twee hele tonen zitten en dus een grote terts. Bij la klein (figuur 2.9) zit er tussen de la en de do maar anderhalve toon tussen.



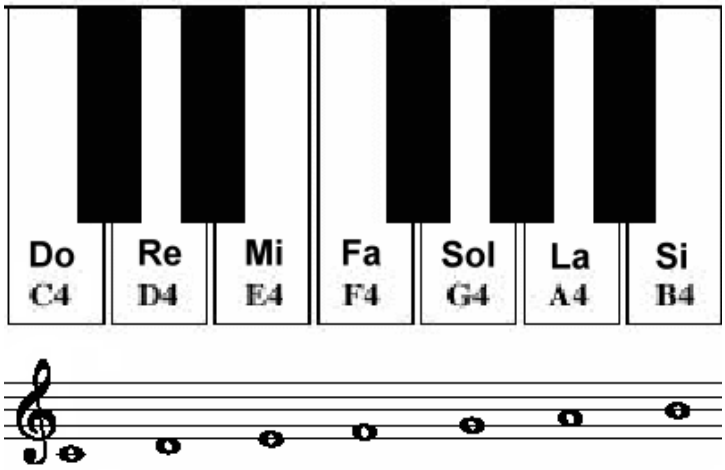
Figuur 2.8: Do groot



Figuur 2.9: La klein

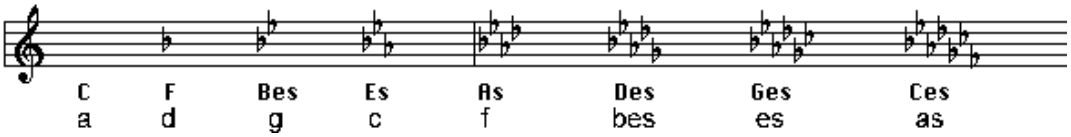
Op de piano (figuur 2.10) is het ook gemakkelijk te zien dat er twee zwarte toetsen tussen liggen bij de do en de mi, tegenover één zwarte toets tussen de la en de do.

[1] Een ander toonsysteem is de chromatische toonladder die alle halve tonen gebruikt.

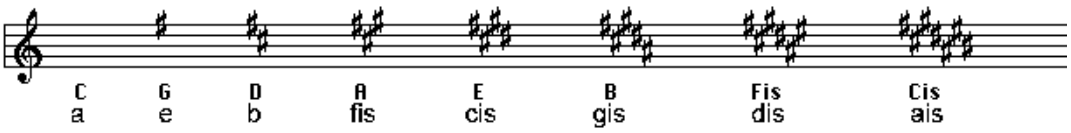


Figuur 2.10: Piano

De andere toonladders (zie figuren 2.11 en 2.12) gedragen zich op dezelfde manier, maar dan zijn er ofwel mollen ofwel kruizen bij de basisnoten, die men op elke nieuwe notenregel vooraan zet, zodat men kan zien dat die bij de toonladder horen en dat de noten niet toevallig verhoogd of verlaagd worden. De grote toonladders worden aangeduid door een grote letter overeenkomstig met de grondtoon van de toonladder. Dit is de hoofdtoon van de toonladder, waar de toonladder naar genoemd is. Bijvoorbeeld, do is de grondtoon van do groot en van do klein. Bij de kleine toonladders wordt dit weergegeven door de kleine letter overeenkomstig met zijn grondtoon.



Figuur 2.11: De voortekening van de toonladders met mollen



Figuur 2.12: De voortekening van de toonladders met kruizen

In een muziekstuk kan de muziek ook van toonladder en toonaard veranderen. Dit noemt men *moduleren*. Het stuk begint en eindigt dan in de basistoonladder,

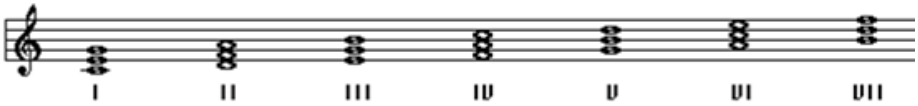
maar verrijkt de melodie onderweg met klanken uit andere toonladders, want elke toonladder heeft zijn eigen klankkleur.

2.6 Homofonie en polyphonie

In de muziekgeschiedenis zijn er twee grote stromingen geweest i.v.m. de theoretische leer van de muziek. Eerst was er de *melodieleer* en nadien de *functieleer*. De melodieleer gaat over het bepalen van een melodie. Een melodie is een veelheid van tonen, samengesteld als klankreeksen, die beantwoorden aan een aantal eisen [2]. Deze eisen verschillen echter van tijd tot tijd en het is niet de bedoeling om hier een vergelijking te maken. Deze melodieleer werd *contrapunt* genoemd, naar het oude "Punctus contra Punctum", hetgeen betekent "punt tegen punt" of in muzikale zin "noot tegen noot" waaruit dan later de betekenis van "melodie tegen melodie" groeit [5]. De Middeleeuwse theoretici hadden het dan ook altijd over *tweeklanken*, het gelijktijdig klinken van twee noten, ook al was dit in tegenstelling tot de meerstemmige polyfone stukken, waar er duidelijk meer noten tegelijkertijd klonken. In de vroegere muziek werden deze melodien echter afgeleid van één hoofdmelodie, de *cantus firmus*. Na de Middeleeuwen werd dit door de nieuwere *polyfone* stukken echter duidelijk, dat men rekening ging houden met meerdere stemmen die samen klonken. Zarlino beschouwde contrapunt als een samengaan of een afspraak, geboren uit het lichaam van de verschillende delen, van de verschillende melodieën, onderverdeeld in verschillende stemmen, die zich scheiden in harmonische intervallen[6]. *Polyfonie* duidt de meerstemmigheid aan van muziekstukken. De verschillende stemmen hebben een bepaalde zelfstandigheid van elkaar. In tegenstelling met de *homofonie* waar er één hoofdstem is, begeleid door de andere stemmen.

Rameau[7] was de eerste theoreticus die de *functieleer* ging beschrijven. Hij baseerde zich op de harmonische begrippen van Zarlino[8], die het belang van tertsen beschreef. Rameau heeft vastgesteld dat er drie tonale functies bestaan. De *tonica*, die de grondtoon van de toonladder bevat, bevindt zich op de 1ste *trap* en zorgt voor een rustpunt. De *subdominant* wekt een spanning op en bevindt zich op de vierde trap van de toonaard, leidend naar de grootste spanning, de *dominant* (de vijfde trap). Hierop kan men dan een *akkoord* bouwen. Een akkoord is het samenklinken van verschillende tonen. Dit gebeurt in de *harmonie* via een opbouw van tertsen. De *grondtoon* die bij een bepaalde trap hoort, is de toon waarop de tertsen gebouwd worden om een akkoord te vormen. Zo bestaat een drie-klank uit zijn grondtoon, een terts en een kwint t.o.v. die grondtoon. Dit kan ook gezien worden als de grondtoon, een terts hoger en nog een terts hoger. Dit kan men nog verder zetten met de vier-klanken, ook wel septiem-akkoorden genoemd, maar dit zou ons te ver leiden. De grondtoon die bij een bepaalde trap horen, werd door Rameau "basse fondamentale" genoemd, omdat die zo belangrijk is. Het is namelijk de basis van de harmonie. Met "La mélodie naît de l'harmonie" (de melodie wordt geboren door de harmonie)

wil hij dat belang benadrukken. Buiten de drie reeds benoemde trappen, zijn er nog neventrappen, die afgeleid zijn van de drie tonale functies. Zo is de 2e graad een afleiding van de vierde graad. Voor de toonladder van do groot kan je in figuur 2.13 de verschillende akkoorden zien, waarvan de bijhorende trap met een Romeins cijfer wordt aangeduid.



Figuur 2.13: De drieklanken

Tenslotte kan men een onderscheid maken tussen de drieklanken.

- Een *grote drieklank* bestaat uit een grote terts en een reine kwint. Op figuur 2.13 zijn de drieklanken van de eerste, vierde en vijfde trap groot.
- Een *kleine drieklank* bestaat uit een kleine terts en een reine kwint. Op de figuur zijn de drieklanken van de tweede, derde en zesde trap klein.
- Een *verminderde drieklank* bestaat uit een kleine terts en een verminderde kwint. Op figuur 2.13 is de drieklank van de zevende trap verminderd.
- Een *overmatige drieklank* bestaat uit een grote terts en een overmatige kwint. Zonder wijzigingstekens komt deze drieklank niet voor.

De grootte van de drieklanken en de bovenvernoemde trappen zijn geldig in elke grote toonladder. In een kleine toonladder is de tweede trap een verminderde drieklank, de eerste, vierde en vijfde een kleine drieklank, en de rest een grote.

Hoofdstuk 3

Inleiding tot CLP

> Constraint Programming represents
> one of the closest approaches
> computer science has yet made
> to the Holy Grail of programming:
> the user states the problem,
> the computer solves it.
> *E. Freuder*

In ons leven krijgen we vaak te maken met beperkingen. Je kan als voorbeeld niet alles kopen met het geld dat je verdient. Ook in het componeren van muziek heb je regels, die ervoor zorgen dat wat je maakt zich beperkt tot een bepaalde stijl van muziek. Hoe we de computer deze regels laten verwerken, wordt in dit hoofdstuk besproken.

3.1 Het paradigma

Constraint Logic Programming of kortweg CLP is een krachtig paradigma dat meer en meer gebruikt wordt als model en om een oplossing te geven aan vele moeilijke problemen, die in ons leven bestaan[9]. Het concept bestaat uit het declaratief beschrijven van regels waaraan de variabelen zich moeten houden. Men kan CLP beschouwen als een generalisatie van het *logisch programmeren* (LP), waar men de unificatie van LP vervangt door een krachtigere probleem oplossend mechanisme[10].

Wat CLP kan, is niet zomaar te bekomen door een paar eenvoudige aanpassingen te doen aan LP-systemen. Predikaten in logisch programmeren kunnen namelijk niet zinvol bekeken worden als regels[11]. Als voorbeeldje nemen we de optelling in de verzameling van de natuurlijke getallen, waar de natuurlijke getallen worden uitgedrukt door de opvolgingsfunctie `s`, zodat het getal `n` wordt voorgesteld door `s(s(s(...(0)...)))` met `n` voorkomens van `s`.

Listing 3.1: Het optellen van natuurlijke getallen

```
plus(0,N,N).
% s(N) is de opvolger van N,
% s(0) is vertaald naar de natuurlijke getallen het getal 1
plus(s(N),M,s(L)) :- plus(N,M,L).
```

Het predikaat `plus(n,m,l)` (zie code 3.1) kunnen we voorstellen als de wiskundige relatie $n + m = l$. Via de logica is het dan gemakkelijk te bewijzen dat de combinatie `plus(N,M,L),plus(N,M,s(L))` geen oplossing zal geven voor variabelen `N`, `M`, `L`. Doordat het logisch programmeren geen globale test doet over de satisfactie van de twee plus-beperkingen, zal het programma in een oneindige zoekactie belanden om toch maar een oplossing te geven. CLP zal de tegenspraak in de regels ontdekken en snel weergeven dat er aan het *satisfactieprobleem* niet voldaan is. Het probleem van satisfactie handelt over het al dan niet bestaan van een oplossing voor het probleem.

Vertrekkende van deze basis werken we de verschillende onderdelen van CLP verder uit. Een belangrijk deel hiervan is de domeinbepaling. Daarna volgt er een uitleg over de beperkingen van een fysisch systeem, om tenslotte te komen tot oplossingen. Om het hoofdstuk af te sluiten wordt er nog een voorbeeld gegeven ter vergelijking van de efficiëntie van een logisch en een CLP-programma.

3.2 Domeinen

Met het *domein* van een probleem bedoelt men het domein waarin de waardes van de variabelen zich kunnen bevinden. Zo kunnen de variabelen bijvoorbeeld komen uit het domein van de *rationale getallen*, *reeële getallen*, van de *boole variabelen*, van verschillende *intervallen* of uit een *eindig domein*. In deze laatste categorie komen de waardes van de variabelen uit een eindige verzameling van gehele getallen. In dit domein zitten de *constraint satisfaction problemen*, zoals het N-koninginnenprobleem en het kleuren van een map, waar men voor de variabelen waarvan de waarde zich in een eindig domein bevinden aan de hand van een aantal regels, een waarde toekent die niet in strijd is met deze regels. Een specifieke categorie hiervan is die van de *planners*, met o.a. uurroosters of opdrachten.

In SICStus Prolog kan er om het domein te initialiseren, gebruik gemaakt worden van:

- `X in 1..10`: X hoort tot de verzameling 1 tot en met 10
- `Y in {1,3,5,7,9}`: Y is 1 of 3 of 5 of 7 of 9

- `Z in {1,3}\/{7,10}`: Z is 1 of 2 of 3 of 7 of 8 of 9 of 10
- `R in \{3,4,5}`: R is niet 3 of 4 of 5
- `domain([X1,X2,X3],0,15)`: De variabelen X1, X2 en X3 behoren tot de verzameling van 0 tot 15

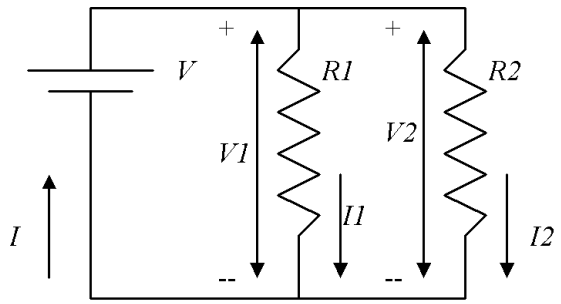
In principe kan je altijd oneindig hoog/laag gaan in je muzieknotatie. In de praktijk zijn menselijke stemmen (of die van muziekinstrumenten) beperkt. Daarom hebben we in het programma gekozen om eindige domeinen te gebruiken om noten voor te stellen.

3.3 Beperkingen (constraints)

Een *beperking* is een voorwaarde op een aantal variabelen die het aantal oplossingen voor deze variabelen verkleint. Beperkingen kunnen daardoor ook gebruikt worden om domeinen van variabelen te verkleinen. Beperkingen worden gebruikt om het gedrag van een systeem van objecten in de realiteit te modelleren door een geïdealiseerd beeld van de interactie tussen de verschillende objecten. Deze simplificatie van de werkelijkheid kan ervoor zorgen dat we die zelfde werkelijkheid beter zouden begrijpen. Op die manier kunnen bepaalde gedragingen van het fysische systeem voorspeld worden. De moeilijkheid bij deze idealisering is het evenwicht te vinden voor een goede abstractie. Te veel abstractie zou de verbinding met de werkelijkheid verliezen, te weinig abstractie zou de beperkingen onbegrijpbaar maken[12].

Een gekend voorbeeld waar men in de fysica abstractie maakt zijn de wet van Ohm en de wetten van Kirchhoff. Aan de hand van het stroomschema (figuur 3.1) kunnen de volgende beperkingen worden afgeleid:

- $V_1 = I_1 * R_1$: Wet van Ohm voor weerstand 1
- $V_2 = I_2 * R_2$: Wet van Ohm voor weerstand 2
- $I - I_1 - I_2 = 0$: 1ste wet van Kirchhoff
- $-I + I_1 + I_2 = 0$: 1ste wet van Kirchhoff
- $V - V_1 = 0$: 2de wet van Kirchhoff
- $V - V_2 = 0$: 2de wet van Kirchhoff
- $V_1 - V_2 = 0$: 2de wet van Kirchhoff



Figuur 3.1: Een eenvoudig stroomschema

Aan de hand van deze beperkingen kan het volledige gedrag van het stroomschema gemodelleerd worden. Door bijkomende informatie over een paar variabelen kunnen de andere variabelen berekend worden.

Behalve deze eenvoudige rekenkundige beperkingen kunnen er ook beperkingen uit de *propositielogica* komen zoals `#\/` de logische OR-functie weergeeft. *Combinatorische* beperkingen zijn een derde groep, die wat veelvoorkomende complexere problemen aanpakt. Een voorbeeld hiervan is `all_different(lijst)` die er voor zorgt dat alle elementen in de lijst verschillend van elkaar moeten zijn.

3.4 De probleemoplosser (solver)

De *probleemoplosser* is de kern van CLP. Het wordt gebruikt als een zwarte doos, waar we de regels aan geven, waarop de probleemoplosser een antwoord gaat geven. Bij de constraint satisfaction problemen zijn er echter twee verschillende vragen die men kan stellen in verband met de beperkingen. Een eerste handelt over het bestaan van mogelijke oplossingen, de tweede wil een oplossing weten. Deze twee vragen kunnen natuurlijk duidelijk in verband met elkaar gebracht worden. Als je een oplossing gekregen hebt, dan is ook het existentieprobleem opgelost. Er zijn echter problemen waarvan men weet dat de oplossing bestaat (er bestaat de kleinste universele Turing-machine), maar waar de specifieke oplossingen nog niet gevonden zijn.

Om dit satisfactieprobleem op te lossen, kan men de waardes van de variabelen over het gehele domein één voor één gaan bekijken of er aan alle beperkingen is voldaan. Als dit gebeurt in eindige domeinen, dan kan men alle oplossingen nagaan. Als het over rationale getallen gaat, is dit niet meer mogelijk. Een techniek die men hier kan toepassen, is het omzetten van de vergelijkingen van de beperkingen naar andere vergelijkingen, waarvan bekend is, dat ze oplossingen geven of niet. Deze simplificatie van het probleem is natuurlijk ook nuttig in domeinen waar men wel alle oplossingen kan uitproberen, om zo de efficiëntie van de probleemoplosser te verhogen.

Voor sommige problemen wil men niet zomaar een oplossing krijgen en zoekt men naar de *beste* oplossing[13]. Zo kan de ene melodie mooier klinken dan de andere, ook al werden dezelfde regels gebruikt. Door optimalisatie kan men dit proberen te bekomen. Deze problemen noemt men soms *Constraint Satisfactie Optimisatie Problemen*[14].

Als je een probleem met beperkingen door zo'n probleemoplosser stuurt, dan krijg je standaard een antwoord op het satisfactieprobleem. In de meeste CLP-systemen zit er gelukkig nog een backtrack-mechanisme dat ervoor zorgt dat je de mogelijke oplossingen dan ook te zien krijgt. Dit wordt meestal door een ingebouwd predikaat gedaan, met de Amerikaans-Engelse benaming *labeling*.

3.5 Voorbeeld

Als we alles nu achter elkaar zetten krijgen we het volgende schema voor een klassiek CLP-programma:

1. *Generatie* van de variabelen binnen een bepaald domein
2. Verzameling van beperkingen
3. *Toekenning*(labeling) van waarden aan de variabelen

Om een vergelijking te trekken met logisch programmeren nemen we het klassieke logische *Send More Money*-probleem. Het probleem bestaat er in dat de 3 woorden (Send, More, Money) getallen voorstellen, met elke verschillende letter een verschillend getal en waarbij je door de optelling van het getal `Send` en het getal `More` bij het getal `Money` moet uitkomen. In voorbeeld 3.2 genereert men telkens een mogelijke oplossing en test men nadien op de volgende beperkingen of dat het een oplossing voor het probleem is. Dit geeft 1.814.400 combinaties die overlopen moeten worden. Er zijn efficiëntere programma's te schrijven met het klassieke Prolog, maar deze versie heeft een algoritme dat niet helemaal naïef is, maar het probleem toch zo eenvoudig mogelijk implementeert, zonder dat men een bepaalde oplossingsmethode suggereert om Prolog op weg te helpen.

In code 3.3 wordt er eerst het domein bepaald, dan het domein beperkt en dan een oplossing gegenereerd, zoals het in het klassieke schema wordt weergegeven. Als men de `constraint([S,E,N,D,M,O,R,Y])` zou omwisselen met `labeling([],[S,E,N,D,M,O,R,Y])`, dan krijgen we dezelfde efficiëntie als in de klassieke versie, omdat men dan via labeling al het backtrack-mechanisme gebruikt wordt voor dat de beperkingen bekeken worden. Hierop zal het programma het hele domein overlopen zonder dat de beperkingen het domein eerst hebben verkleind.

Listing 3.2: Send More Money: Prolog-versie

```
% send + more = money
smm([S,E,N,D,M,O,R,Y]) :-
        Digits     = [0,1,2,3,4,5,6,7,8,9],
         assign_digits  ([S,E,N,D,M,O,R,Y], Digits),
        M > 0,
        S > 0,
                    1000*S + 100*E + 10*N + D +
                    1000*M + 100*O + 10*R + E =:=
        10000*M + 1000*O + 100*N + 10*E + Y.

 select (X, [X|R], R).
 select (X, [Y|Xs], [Y|Ys]):- select (X, Xs,Ys).

  assign_digits   ([],   _List ).
  assign_digits  ([D|Ds], List):-
         select (D, List , NewList),
          assign_digits  (Ds, NewList).
```

Listing 3.3: Send More Money: CLP-versie

```
% send + more = money

:- use_module( library ( clpfd )).

smm(S,E,N,D,M,O,R,Y) :-
      domain([S,E,N,D,M,O,R,Y],0,9),
       constraint  ([S,E,N,D,M,O,R,Y]),
      % labeling :  waarde toekennen  via   backtracking
       labeling  ([],[ S,E,N,D,M,O,R,Y]).

 constraint  ([S,E,N,D,M,O,R,Y]) :-
      S #\= 0, M #\= 0,                  % geen nullen  van  voor
      % De  verschillende    letters    moeten  verschillende    waarden hebben
        all_different   ([S,E,N,D,M,O,R,Y]),
                    1000*S + 100*E + 10*N + D
      +             1000*M + 100*O + 10*R + E
      #= 10000*M +1000*O + 100*N + 10*E + Y.
```

Hoofdstuk 4

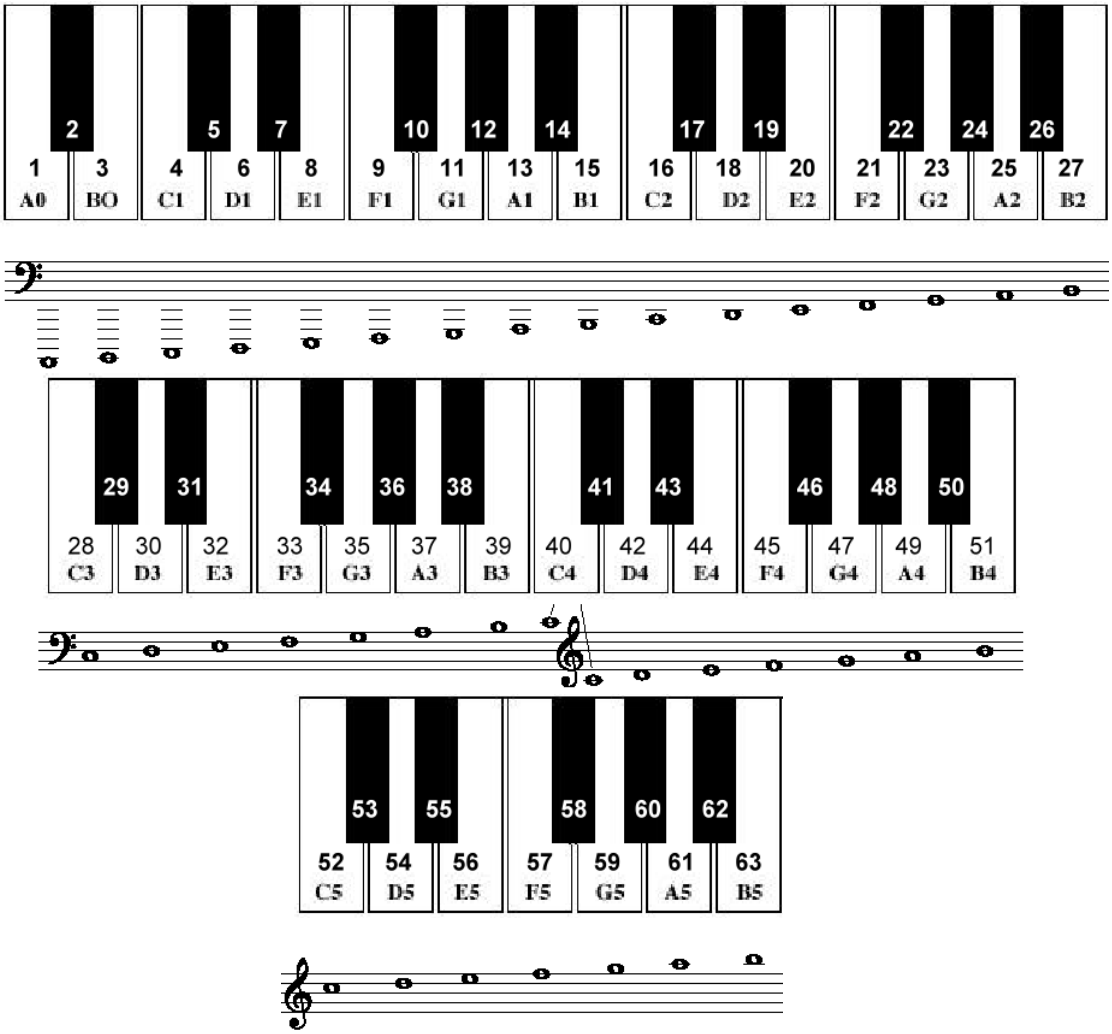
Kennisrepresentatie van de muziek

Hoe we de fundamenten van de muziek voorstellen in het programma bespreken we in dit hoofdstuk. Dit is over het algemeen een numerieke notatie, omdat we gebruik maken van een CLP-systeem voor eindige domeinen en die dus bestaat uit een eindige set van gehele getallen.

4.1 Representatie van noten

De belangrijkste bouwsteen in de muziek is een toon. Een toon wordt gekenmerkt door enerzijds zijn frequentie en anderzijds zijn lengte. Naar analogie van de muzikaal symbolische notatie van een toon, wordt er in het programma gesproken over het functiesymbool *noot* met een ariteit van 2. De argumenten van *noot* zijn hoogte en lengte,die met een geheel getal worden voorgesteld. Zo wordt de lengte van een hele noot voorgesteld als 1, die van een halve als 2, een vierde als 4, en zo verder. Om de toonhoogte te bepalen waren er twee mogelijkheden, een absolute en een relatieve. De absolute toonhoogte is een getal, dat in elke melodie dezelfde noot zal aanduiden. De relatieve toonhoogte kan echter rekening houden met de toonladder en de melodie. Het programma volgt de absolute bepaling van de toonhoogte. Dit is de eenvoudigste manier van voorstellen, omdat een bepaalde toon in het programma altijd op dezelfde manier zal aangeduid worden. De ondergrens van het domein (1) komt overeen met de laagste noot dat op een piano kan gespeeld worden (a0). Het grootste getal (88) stelt op zijn beurt de hoogste noot op een piano voor (c8). Figuur 4.1 stelt de meest gebruikelijke noten voor. Samenvattend in een voorbeeld stelt `noot(40,4)` een vierde noot voor met de toonhoogte van c4.

De noten horen bij een bepaalde stem. Deze stem wordt voorgesteld als een lijst van noten. Omdat het programma op dit ogenblik geen rekening houdt met ritmes, worden de stemmen via de hoofdmodules (contrapunt/harmonie) doorgeven als een lijst van toonhoogtes. Bij uitbreiding van de regels, kan dit gemakkelijk aangepast worden omdat men enkel in de nieuwe modules die handelen over een variatie van



Figuur 4.1: De toonhoogtes voorgesteld op een piano

Grote toonladder	**Grondnoot**	**Kleine toonladder**	**Grondnoot**
C	4	a	1
mollen			
F	9	d	6
Bes	2	g	11
Es	7	c	4
As	12	f	9
Des	5	bes	2
Ges	10	es	8
Ces	3	as	12
kruisen			
G	11	e	8
D	6	b	3
A	1	fis	10
E	8	cis	5
B	3	gis	12
Fis	10	dis	7
Cis	5	ais	2

Tabel 4.1: Afspraken omtrent de toonladder

ritme de noten met hoogte en lengte moet doorgeven.

4.2 Representatie van toonladders, akkoorden en intervallen

Een toonladder wordt volledig bepaald door de grondnoot en de grootte van de toonaard. (Als men enkel de grondnoot zou gebruiken, zou men geen verschil kunnen maken tussen do groot of do klein.) Om de grondnoot eenduidig te bepalen, wordt die voorgesteld door het kleinste voorkomen van die noot op een piano (88-toetsen, zoals in figuur 4.1). In tabel 4.1 zie je de voorstelling van de grondnoten. Een kleine toonaard wordt voorgesteld door een 1 en een grote door een 2. Als voorbeeld wordt do groot voorgesteld door hoogte 4 en grootte 1. Een toonladder is dus voorgesteld door een paar `(grondnoot,grootte)`.

In de muziek is het gebruikelijk om tijdens een compositie af en toe van toonaard en toonladder te veranderen. Dit implementeren zou echter te veel mogelijkheden creëren. In plaats daarvan is geopteerd, dat bij de invoer ook de modulaties[1] zouden worden gegeven. Deze implementatie viel echter buiten het tijdsbestek van de thesis. Het programma zal dus enkel composities genereren, die geen modulaties hebben.

[1] veranderen van toonaard

Akkoorden worden weergegeven door hun functie. Deze functie wordt voorgesteld als een geheel getal en komt overeen met de trappen in figuur 2.13. In het programma komt men dus de lijst `[akkoorden]` tegen.

Intervallen worden weergegeven door de positieve afstand van 2 noten. De waarde van de laagste noot wordt afgetrokken van de waarde van de hoogste noot en zo bekomt men die afstand. Deze intervallen komen binnen de polyfonie voor als de lijst `[intervallen]`.

Hoofdstuk 5

Homofonie

De muziek is de wiskunde voor het gevoel.
De wiskunde is de muziek der rede.
J.J.Sylvester

5.1 Inleiding

Een homofoon stuk is een compositie die bestaat uit een hoofdmelodie en een daarbij horende begeleiding. Die begeleiding bestaat uit een opeenvolging van akkoorden. Daarom wordt de homofonie ook wel als synoniem gebruikt voor de harmonieleer, wat de leer van de relaties tussen de akkoorden is. Ten tijde van Pythagoras was er reeds van harmonie sprake. Zij vonden dat iets in harmonie was als men het in getallen kon uitdrukken. Zo ook vonden ze het mysterie van de muziek verborgen in de verhouding van de getallen. Deze sterke verbintenis tussen wiskunde en muziek wordt ook vertaald door de vele wiskundigen, die muziek componeren, zoals Euler en Pythagoras en door de componisten, die de muziek wiskundig gingen beschrijven, zoals Zarlino en Rameau.

Als muziektheorie is de harmonieleer in de zestiende eeuw zelfstandig geworden. Waar men daarvoor horizontaal keek, stem per stem, kijkt men in de harmonie verticaal. Men kijkt naar de hoofdmelodie en men plaatst er een akkoord onder. Op deze wijze zijn de onderliggende stemmen volledig afhankelijk van de oorspronkelijke stem. Het belangrijkste werk in verband met de klassieke harmonieleer is geschreven door Jean Philippe Rameau in de achtiende eeuw. Geïnspireerd door de regels van Rameau en door de regels die worden aangeleerd in de muziekacademie, zijn er in het programma een set van beperkingen geïmplementeerd. Men kan dit zien als een eenvoudige harmonieleer, die niet volledig is, maar toch voldoende om mooie composities te beschrijven.

In de loop der tijden zijn er dus door diverse theoretici geschriften over harmonie geschreven. Opvallend is dat de ene theoreticus een re-

> gel heel anders formuleert dan de andere theoreticus. Soms beweren zij over hetzelfde onderwerp precies het tegenovergestelde. Dit bewijst dat ieders visie persoonlijk is. Typerend is echter wel dat de componisten er zich niets van aantrokken. Voor hen waren wetenschappelijke overwegingen van weinig of geen waarde. [15]

5.2 Beperkingen

5.2.1 Akkoorden

In hoofdstuk 2.6 hadden we het al over Rameau die als eerste de 3 belangrijkste akkoorden in een toonaard beschrijft. Dit zijn de tonica (of grondakkoord), de onderdominant (of subdominant) en de bovendominant (of dominant). Alledrie zijn het drieklanken. Dit wil zeggen dat het akkoord uit 3 noten bestaat, waarvan er tussen de grondtoon van het akkoord en de 2 andere noten zich respectievelijk een terts[1] en een kwint bevinden.

Tonica: De grondtoon van de tonica is de grondnoot van de toonladder. Als voorbeeld is de onderste noot van de tonica in do groot een do.

Dominant: Deze drieklank bevindt zich een kwint hoger dan de tonica.

Subdominant: Tenslotte bevindt de subdominant zich een kwint lager dan de tonica.

Deze akkoorden zijn in een grote toonaard grote akkoorden. Dit wil zeggen dat de intervallen tussen de grondtoon en de 2 andere noten respectievelijk een grote terts en een reine kwint zijn. De eenvoudige harmonieleer die het programma gebruikt bevat ook de nevendrieklanken van deze akkoorden. De nevendrieklanken zijn de drieklanken, die op de neventrappen gebaseerd zijn, dus op de tweede, de derde en de zesde trap. Ze zijn afhankelijk van hun grote broertjes respectievelijk de akkoorden van de vierde, de vijfde en de eerste trap. Zelf zijn het kleine akkoorden. Een akkoord is klein als de intervallen tussen de noten zich verhouden als kleine terts en reine kwint. De graden van de akkoorden worden voorgesteld door Romeinse cijfers.

I De tonica

II De boventonica

III De mediant

IV De subdominant

[1] zie tabel 2.1 voor meer informatie over intervallen

V De dominant

VI De submediant

VII De leidtoon

Deze akkoorden worden soms anders genoemd, maar worden altijd met hun graden benoemd. De nevendrieklanken zijn afhankelijk van de hoofddrieklanken. Zo wordt het III-de graadsakkoord gezien als een verzwakte dominant. De boventonica is dan op zijn beurt ook een verzwakte subdominant. Tenslotte is de VI-graad een verzwakking van de tonica. Het leidtoonakkoord is niet opgenomen in de eenvoudige harmonieleer, omdat dit in een grote toonladder een verkleinde drieklank is. Dit wil zeggen dat de intervallen van de tonen in de drieklank zich verhouden als kleine terts en als verminderde kwint.

Deze akkoorden hebben ook een bepaalde functie. Zo zet de tonica de toonaard van de toonladder en domineert de dominant het hele muziekstuk. De dominant wekt namelijk een bepaalde spanning op, doordat het de 2 tonen bevat, die rond de grondnoot liggen, om zich op te lossen naar de tonica. Deze oplossing kan echter uitgesteld worden door het toevoegen van een VI-de graadsakkoord. Dit kan dan opgelost[2] worden naar de tonica, daar alles kan opgelost worden naar dit akkoord, maar ook naar de subdominant. Zo zijn er bepaalde *opvolgingsregels*, die bepalen welke akkoorden er kunnen volgen op andere akkoorden:

- De tonica mag gevolgd worden door elk ander akkoord
- De boventonica mag niet gevolgd worden door de submediant
- De mediant mag niet gevolgd worden door de submediant en de boventonica
- De subdominant mag niet gevolgd worden door de mediant
- De dominant mag niet gevolgd worden door de boventonica en ook niet door de subdominant
- De submediant mag niet gevolgd worden door de boventonica, de mediant en de dominant

Als voorbeeld van de implementatie is in listing 5.1 de opvolging van de dominant beschreven.

Een andere beperking houdt zich bezig met het *einde* van een compositie of van een zin. Een volledig stuk zou moeten eindigen op de tonica. Een zin moet eindigen op een hoofd-drieklank. In code 5.2 wordt beschreven, dat het laatste akkoord van een zin de tonica, subdominant of dominant moet zijn.

[2]het volgende akkoord kan een oplossing bieden: spanning verminderen of vermeerderen

Listing 5.1: Opvolgingsregel voor de dominant

```
/*
  opvolgings_regel (Akkoorden).
   akkoorden mogen door sommige andere akkoorden niet gevolgd worden
   ([ integer ])  (in)
 */

opvolgings_regel ([5, Akk2 ]) :- !,
    (#\ (( Akk2 #= 4) #\/ ( Akk2 #= 2))).

opvolgings_regel ([5, Akk2|Rest ]) :- !,
    (#\ (( Akk2 #= 4) #\/ ( Akk2 #= 2))),
    opvolgings_regel ([Akk2|Rest]).
```

Listing 5.2: Een zin eindigt op een hoofd-drieklank

```
/*
   einde_constraint (Akkoorden).
   is waar als het einde van een zin eindigt op de 1ste , 4de of 5de graad
  ( lijst_van_integers ) (in)
 */

einde_constraint (Akkoorden) :- last (Akkoorden,Akkoord), member(Akkoord,[1,4,5]).
```

Listing 5.3: De beperking van de volle akkoorden

```
/*
  akkoorden_volledig(Sopraan,Bas,Alt,Tenor) :-
    Er zijn 3 verschillende noten in de 4 stemmen.
  ( lijst_van_integers , lijst_van_integers , lijst_van_integers , lijst_van_integers )
  (in,in,in,in)
  */

  akkoorden_volledig  ([],[],[],[]).

  akkoorden_volledig ([S_noot|Rest_s ],[B_noot|Rest_b ],[A_noot|Rest_a ],[T_noot|Rest_t ]) :-
    % De noten worden eerst herleid naar hun laagste voorkomen op de piano
    mod12(S_noot,Sopraan),
    mod12(T_noot,Tenor),
    mod12(A_noot,Alt),
    mod12(B_noot,Bas),
    % ofwel zijn sopraan, bas en alt zijn verschillend
    ((( Sopraan #\= Bas ) #/\  (Sopraan #\= Alt ) #/\  (Bas #\= Alt )) #\/
    % ofwel zijn sopraan, tenor en alt zijn verschillend
     (( Sopraan #\= Tenor ) #/\  (Sopraan #\= Alt ) #/\  (Tenor #\= Alt )) #\/
    % ofwel zijn sopraan, tenor en bas zijn verschillend
     (( Sopraan #\= Bas ) #/\  (Sopraan #\= Tenor ) #/\  (Bas #\= Tenor )) #\/
    % ofwel zijn tenor, bas en alt zijn verschillend
     (( Tenor #\= Bas ) #/\  (Tenor #\= Alt ) #/\  (Bas #\= Alt ))),
     akkoorden_volledig(Rest_s,Rest_b,Rest_a,Rest_t).
```

Een akkoord klinkt *voller*, als alle noten van dat akkoord in de 4 stemmen zijn opgenomen. Dit is geen noodzakelijk beperking om een mooie melodie te vormen, maar hoort wel tot de basis van de harmonieleer en zeker als het over drieklanken gaat. Daar de sopraan gegeven is en de bas de belangrijkste ondersteunende stem is, die direct na de akkoorden wordt berekend, worden in het programma de twee overige stemmen samengenomen in één module, zodat zij samen de beperking in verband met de volle akkoorden (listing 5.3) kunnen voldoen. In die module wordt er gebruik gemaakt van `mod12(getal1,getal2)`, waarbij `getal2` het `getal1` modulo 12 uitdrukt, met als enige verschil dat hij i.p.v. een getal tussen 0 en 11 te geven, de 0 herleidt naar 12.

5.2.2 Stemafhankelijke beperkingen

In de eerste plaats zijn er natuurlijk de *begrenzingen* van de verschillende stemmen. Zo ligt het bereik van een sopraan tussen de lage do en de hoge la (Dit is volgens

Listing 5.4: Domeinbepaling van de stemmen

```
/*
    In de module voegBasToe.pl
 */

 domain(Bas,21,42),

/*
    In de module voegTenorAltToe.pl
 */

 domain(Alt,33,54),
 domain(Tenor,28,49)
```

Listing 5.5: De stemmen starten op een comfortabele noot

```
 /*
   begin_constraint (Bas) :- de eerste noot van de bas zit tussen de si en de sol
  ( lijst_van_integers ) ( in )
  */
   begin_constraint ([BasNoot|_]) :-
     BasNoot #>= 27,
     BasNoot #=< 35.
```

figuur 4.1 tussen 40 en 61), die van de tenor ligt daar een octaaf lager van. Het bereik van een alt is van de lage fa naar de hoge re (tussen 33 en 54) en die van de bas ligt daar weer een octaaf lager van.[2]

Via CLP worden de domeinen bepaald met het predikaat domain. Die zorgt ervoor dat de variabelen in de gegeven lijst, in dit geval een lijst met toonhoogtes (Bas, Alt, Tenor), gehele getallen moeten zijn binnen dit domein. (zie listing 5.4)

Om het voor een koor niet te moeilijk te maken, heb ik er voor gekozen, dat de eerste noot van de tenor, alt en bas in een beperkt gebied ligt, dat het beste zingbaar is voor die partijen. Voor de tenor ligt dat gebied tussen e3 (vertaald naar figuur 4.1 wordt dat 32) en d4 (42). De bas begint tussen b2 (27) en g3 (35). Tenslotte begint de alt tussen c4 (40) en a4 (49). Dit wordt in listing 5.5 weergegeven.

Een bijkomende beperking handelt ook over het comfortabel zingen. Het is namelijk niet goed voor de stem om in te lage of te hoge registers te lang te blijven zingen. Daarom zijn er bij elke stem beperkingen gedefinieerd die ervoor zorgen dat de stem niet te lang in de uiterste registers moet blijven zingen.

Listing 5.6: Er moet een kleine afstand zijn tussen noten die achter elkaar komen

```
/*
     kleine_afstand (Noot,Andere_noten) :-
         de afstand tussen de noten zal niet groter dan 3 zijn
    ( integer , lijst_van_integers ) ( in , in )

 */

kleine_afstand (Noot,[StemNoot|RestStem]) :-
    StemNoot #>= Noot - 3,
    Noot #>= StemNoot - 3,
     kleine_afstand ([StemNoot|RestStem]).
     kleine_afstand (_ ,[]).
```

5.2.3 Overige beperkingen

Een algemene beperking in de eenvoudige harmonieleer, die ook voor contrapunt zal gelden, is, dat geen enkele noot in de verschillende stemmen, een *toonladdervreemde* noot is. Dit zijn noten, die niet voorkomen in die toonladder en kunnen bekomen worden door wijzigingstekens aan te brengen. In de eenvoudige harmonieleer wordt dit bekomen, door de beperking in te voeren dat elke noot bij een bepaald akkoord hoort, zoals beschreven is in 5.2.1. Omdat in deze voorgedefiniëerde akkoorden geen toonladdervreemde noten voorkomen is aan deze beperking voldaan.

Omdat elke stem *zingbaar* moet zijn, moet er een bijkomende beperking komen, zodat de noten die op elkaar volgen niet te ver uit elkaar liggen. Deze beperking is algemeen geldig voor alle stemmen en voor zowel harmonie als contrapunt, daar het een beperking is met betrekking tot koormuziek, waarop de thesis gebaseerd is. Daarom is dit in een aparte module beschreven, zodat het gemeenschappelijk gebruikt kan worden. In code 5.6 is er voor gekozen om die afstand te beperken tot 3 halve tonen.

In de praktijk gebeurt het wel vaker dat de ene stem soms een andere stem *kruist*. Hiermee wordt er bedoeld dat een lagere stem hoger gaat zingen dan een hogere stem. Dit mag natuurlijk niet te lang blijven duren, omdat men anders die melodie beter aan de andere stem kan geven. Er is geopteerd om deze mogelijkheid uit te sluiten in het genereren van oplossingen, omdat dit anders te veel extra mogelijkheden zou creëren tijdens de generatie. In listing 5.7 staat de nodige code om deze beperking weer te geven.

Tenslotte is er in het programma rekening gehouden met *variatie*. Het is niet leuk om een melodie te zingen die de ganse tijd op dezelfde toonhoogte blijft hangen. Volgens de theorie van de harmonie is het perfect mogelijk en goed om een stem dit te laten doen. In sommige gevallen gebeurt dit zelfs. Om deze melodiën te verzinnen

Listing 5.7: De stemmen mogen onderling elkaar niet kruisen

```
/*
  niet_kruisen (Hoge_stem,Lagere_stem) :-
    De hoge stem wordt nooit lager dan de lagere stem.
 ( lijst_van_integers , lijst_van_integers ) ( in , in )
 */

  niet_kruisen ([],[]).
  niet_kruisen ([ Hoge_stem_noot|Rest_hoog ],[ Lage_stem_noot| Rest_laag ]) :-
    Hoge_stem_noot #>= Lage_stem_noot,
     niet_kruisen (Rest_hoog, Rest_laag ).
```

afgeleid van een hoofdmelodie is er echter geen al te grote intellectuele inspanning nodig en kan men dit gemakkelijk manueel doen. Als men componeert, houdt men echter rekening met deze beperking, zonder dat dit in een regel is opgenomen. Ook gaat een componist een melodie rijker maken, door afwisseling te brengen in de opeenvolgende akkoorden. Dus eisen we dat van elke drie opeenvolgende noten of akkoorden er tenminste twee verschillend zijn. Via de module variatieConstraint (zie listing 5.8) wordt hier aan voldaan.

5.3 Prolog vs. CLP

Oorspronkelijk is er voor gekozen om alles in CLP te schrijven. Omdat er echter bij akkoorden een bepaalde volgorde of hierarchie heerst en dit ervoor zorgt dat CLP niet ten volle kan benut worden om dat daar de volgorde niet van belang is. Daarom is de module met betrekking tot de akkoorden niet in CLP geschreven.

Met behulp van sommige goed gekozen beperkingen of door middel van een optimalisatiefunctie zou het toch mogelijk zijn om dit volgens het paradigma van CLP te herschrijven, opdat de voordelen in verband met efficiëntie behouden worden. Een andere mogelijke stap zou zijn om er zachte beperkingen aan toe te voegen, waar men een bepaald kost aan gaat geven en er rekening mee houdt dat het programma een totale kost niet mag overschrijden[16].

Voor de andere modules is er wel gebruik gemaakt van CLP. Hier spreken we over de modules die specifiek te maken hebben met stemmen. Dit zijn `voegBasToe` en `voegTenorAltToe`, waar de regels geen prioriteiten hebben en dit dus uitermate geschikt is voor Constraint Logic Programming.

Listing 5.8: De variatieConstraint-module

```
/*
 module declaratie
 */

:- module( variatieConstraint ,[ var_constr /1]).

/*
  var_constr ( Lijst ).
    Er moet voldoende variatie zijn in de Lijst
  ( lijst_van_integers ) (in)
 */

var_constr ([Element1,Element2,Element3 ]) :- !,
    (#\ (( Element1 #= Element2 ) #/\ ( Element1 #= Element3 ) #/\ ( Element2 #= Element3 ))).
var_constr ([Element1,Element2,Element3|Rest]) :-
    (#\ (( Element1 #= Element2 ) #/\ ( Element1 #= Element3 ) #/\ ( Element2 #= Element3 ))),
    var_constr ([Element2,Element3|Rest ]).
```

Hoofdstuk 6

Polyfonie

Als u in de tijd terug wilt reizen,
is muziek de zekerste en veiligste weg.
onbekend

6.1 Inleiding

De *meerstemmigheid* of polyfonie heeft als basis de leer van het contrapunt. Deze leer is een kunst in het schrijven voor zangstemmen, die de perfectie kunnen nastreven in tegenstelling tot instrumenten, die de volmaaktheid van de natuur maar kunnen benaderen[6]. Via deze hypothese legt Zarlino de band tussen zijn rationele theorie over *consonanten* en de muzikale praktijk. Consonanten definieert hij als een mengeling van hoge en lage klanken, die zacht en gelijkmatig de oren strelen en een goed gevoel geven[8]. Dit is in tegenstelling tot de *dissonanten*, die een wrang gevoel geven. Contrapunt bestaat voornamelijk dan ook uit consonanten met in de tweede plaats een aantal dissonanten. In overeenstemming met Plato's melos (lied), die bestond uit logos (wat er met een woord wordt uitgedrukt), harmonia (afspraak of relatie van geluiden) en rhythmos (tijd en ritme)[17], bestond volgens Zarlino een polyfone vocale compositie (melodia) uit 3 dingen. Dit waren 2 of meerdere stemmen die samen bewogen (harmonia), op een bepaald ritme (numero) en op een bepaalde tekst (oratione).

Zarlino lag niet aan de basis van de regels van het contrapunt. Net zoals vele andere theoretici beschreef hij via regels de muziek van de meeste componisten van hun tijd. Het was dus niet de componist die de regels leerde, maar het waren de theoretici die de regels leerden uit de contemporaine composities. Nadien kon men dan via die regels muziek schrijven, zodat het klonk of het oorspronkelijk uit een vroeger tijdperk kwam. Als motivatie voor het schrijven van zijn boeken gaf Zarlino dat goddelijke muziek jammerlijk in verval geraakt is en door sommigen beledigd wordt. Hij prees zijn leermeester Adriaen Willaert, als door God gezonden

om de muziek in ere te herstellen in waardigheid. Zarlino heeft voor ditzelfde ideaal zijn kennis en wetenschap verwoord en beschreven, opdat de muziek weer volmaakt zou worden.

6.2 Beperkingen

6.2.1 Algemeen

Zarlino beschrijft de 6 noodzakelijke voorwaarden waaraan elke compositie aan moet voldoen[6]:

1. Een *onderwerp*: dit is de hoofdmelodie, waaraan de andere stemmen aan worden gespiegeld
2. Voornamelijk *consonanten*: tonen die mooi samen klinken (zie 6.2.2)
3. Goede bewegingen: De stemmen moeten t.o.v. elkaar op een bepaalde wijze bewegen
4. Variatie in bewegingen: het mag niet te eentonig worden
5. 1 (*gregoriaanse*) toonaard: Alle stemmen bevinden zich in dezelfde toonaard
6. Overeenstemming met de tekst (blije tekst/blije muziek)

Aan het eerste puntje is er in onze probleemstelling al voldaan. De gegeven melodie wordt namelijk als onderwerp gekozen. Hierbij moet opgemerkt worden, dat in de tijd van Zarlino het onderwerp zich meestal in de tenorstem zich bevond. Doordat het bereik van de tenor en de sopraan een octaaf verschil heeft, zou je gemakkelijk het onderwerp toch in de tenor kunnen zetten. Dit hebben we toch niet gedaan, omdat men in deze tijd de sopraan als leidstem promoot boven de tenor.

De regels 2 tot en met 4 worden in hoofdstuk 6.2.2 verder uitgewerkt, omdat deze regels het grootste gewicht bepalen van het programma.

In de tekst van Zarlino wordt er rekening gehouden met *kerktoonladders* (gregoriaans). Kort uitgelegd zijn dit sequenties van 8 opeenvolgende witte toetsen op de piano, die bepaald wordt door de *finalis* (dit is de laatste toon van het werk) en door de *reciteertoon*, ook wel *dominant* of *tenor* genoemd, waarop men reciteerde. Na de gregoriaanse toonaarden zijn er de hedendaagse toonladders gekomen (zie 2.5). Door deze laatste toonladders kan men via hun toonaard een impressie geven over de gemoedsgesteldheid van de muziek. Op de vraag of we de hedendaagse toonladders, die we in de harmonie bekeken hebben, mogen gebruiken binnen de leer van het contrapunt, geeft Ernest W. Mulder een positief antwoord. De verworvenheden

Listing 6.1: De noten moeten in de juiste toonaard zitten

```
/*
  module declaratie
  */

:- module(notenInToonaard,[ alle_noten_toonaard /3]).

/*
      alle_noten_toonaard (Noten,GrondNoot,Grootte)
          :- noten zit in de toonaard
     ( lijst_van_integers , integer , integer ) ( in , in , in )
 */

  alle_noten_toonaard ([], _, _).
  alle_noten_toonaard ([Noot|RestNoot],GrondNoot,Grootte) :-
     noten_toonaard(Noot,GrondNoot,Grootte),
      alle_noten_toonaard (RestNoot,GrondNoot,Grootte).

/*
     noten_toonaard(Noot,GrondNoot,Grootte)
          :- Noot zit in de toonaard
     ( integer , integer , integer ) ( in , in , in )
 */

  %grote toonaard
   noten_toonaard(Noot,GrondNoot,2) :-
        Diff #= Noot - GrondNoot,
        DM #= Diff mod 12,
        DM in \{1,3,6,8,10}.

  %kleine toonaard
   noten_toonaard(Noot,GrondNoot,1) :-
        Diff #= Noot - GrondNoot,
        DM #= Diff mod 12,
        DM in \{1,4,6,9,11}.
```

van de homofonie kunnen toegepast worden in de polyfonie als er niet aan de grondbeginselen wordt geraakt van die laatste. Zo mag er nooit één stem prevaleren en nooit mag het harmonische schema met akkoorden op de voorgrond treden[5].

Voor een programma is het moeilijk om overeenstemming met de tekst te gaan vinden. Men zou dit als parameter kunnen meegeven of men moet met tekstherkenning gaan werken.

In het programma worden de laatste twee puntjes dan ook samengenomen en is het voldoende om de toonaard mee te geven om de klankkleur te bepalen. Hoe dat die toonladders intern voorgesteld worden kan je in tabel 4.1 lezen. Via de module notenInToonaard (zie listing 6.1) wordt er voor gezorgd dat de noten allemaal in de opgegeven toonladder zitten. De noten die niet in de toonladder zitten worden via het predikaat `noten_toonaard(Noot,GrondNoot,Grootte)` de noten die niet in de toonladder zitten uit het domein van de veranderlijken geschrapt. Het domein van de stemmen is hetzelfde als bij homofonie (zie listing 5.4). Hierbij wordt er rekening gehouden met het feit dat elke toonladder van een bepaalde toonaard dezelfde intervalverschillen heeft tussen opeenvolgende noten (zie figuren 2.8 en 2.9). Met behulp van de grondnoot van de toonaard zijn alle noten dan bepaald.

Over het algemeen beschrijf ik hier de regels van Zarlino. Andere methodes zijn die van Palestrina of die beschreven zijn door Ernest W. Mulder.

6.2.2 Intervallen

In tegenstelling tot de harmonie gaat het in het contrapunt niet over de akkoorden die de 4 stemmen met elkaar maken. Noot tegen noot worden 2 stemmen gelegd en de intervallen tussen die 2 stemmen zijn van belang. In de polyfonie beschrijft men twee groepen van intervallen. Een eerste groep bestaat uit *consonanten*. Deze intervallen zijn spanningloos of statisch en zorgen voor een zekere muzikale stabiliteit. In deze groep zijn er ook gradaties i.v.m. de perfectie van een interval. De meest perfecte intervallen zijn het octaaf en de priem. Ook de kwart en de kwint worden *perfect* genoemd. De *imperfecte* consonanten zijn de terts en de sext, die tussen de perfecte consonanten en de *dissonanten* liggen. Tenslotte blijven de secunde en de septiem over als dissonanten. Enkel over de kwart zijn er meningsverschillen, zodat men die soms bij de *imperfecte* of dissonanten gerekend wordt. Zarlino lost dit op door de kwarten slechts toe te laten bij drie- of meerstemmige muziek. Omdat het in het programma gaat over vierstemmige muziek is de kwart ook in code 6.2 opgenomen. Ook lijkt het logisch om de kwart, de kwint en het octaaf samen te nemen, daar zij ook als rein beschouwd worden in tegenstelling tot de andere.

In het programma maak ik gebruik van het simpele contrapunt. In deze vorm van contrapunt komen geen dissonanten voor. Voor de uitbreidbaarheid is er toch een predikaat i.v.m. consonanten opgenomen in het programma, die de imperfecte en de perfecte bundelt.

Listing 6.2: De perfecte consonanten

```
/*
  perfecte_consonant (Noot1,Noot2)
     :- de afstand tussen noot1 en noot2 is een perfecte consonant
   ( integer , integer ) ( in , in )
 */

  perfecte_consonant (Noot1,Noot2) :- Noot2 #= Noot1.
  perfecte_consonant (Noot1,Noot2) :- reineKwint(Noot2,Noot1).
  perfecte_consonant (Noot1,Noot2) :- reineKwart(Noot2,Noot1).
 % ook octaven hoger horen hier bij
  perfecte_consonant (Noot1,Noot2) :-
     Noot #= Noot1-12, Noot #> Noot2, perfecte_consonant(Noot,Noot2).
```

Listing 6.3: De imperfecte consonanten

```
/*
  imperfecte_consonant (Noot1,Noot2)
     :- de afstand tussen noot1 en noot2 is een imperfecte consonant
   ( integer , integer ) ( in , in )
 */

 imperfecte_consonant (Noot1,Noot2) :- terts (Noot2,Noot1).
 imperfecte_consonant (Noot1,Noot2) :- sext (Noot2,Noot1).
% ook octaven hoger horen hier bij
 imperfecte_consonant (Noot1,Noot2) :-
     Noot #= Noot1-12, Noot #> Noot2, imperfecte_consonant(Noot,Noot2).
```

6.2.3 Het toevoegen van de tenor aan de gegeven sopraanstem

De eerste regels hebben betrekking tot het toevoegen van de eerste stem bij een gegeven melodie. In ons geval is dit dus het toevoegen van de tenor aan de sopraanstem.

Een *eerste* regel is dat een compositie voornamelijk uit consonanten moet bestaan. Een dissonant wordt als versiering gebruikt en is in het programma niet opgenomen geworden.

Een *tweede* regel is dat een compositie moet beginnen met een perfecte consonant. In tegenstelling tot de meeste componisten vond Zarlino dat deze regel zeker niet absoluut was. Men moet geen perfectie vanaf het begin nastreven, maar naar perfectie op het einde. Dit is dan ook de *derde* regel, waar men naar de zuiverste perfectie streeft in de vorm van het octaaf. In het programma worden deze regels allebei absoluut gebruikt.

Een *vierde* regel handelt over de goede afwisseling tussen perfecte en imperfecte consonanten. Omwille van de tweede en de derde regel kan men dit echter niet volhouden als er een even aantal noten in een melodie zijn. Omdat we de tweede regel toch streng hebben gehouden, hebben we ervoor gekozen dat een imperfecte consonant zich altijd moet oplossen naar een perfecte, maar dat dit niet omgekeerd ook zo zou moeten zijn.

Een *vijfde* regel zorgt ervoor dat de stemmen niet te lang dezelfde beweging maken. Als de stemmen teveel parallel met elkaar zouden zijn, dan zou het gevaar bestaan, dat men die stemmen hoort als één stem.

Als *zesde* regel mag een stem niet te veel springen. Deze regel is identiek met een homofonisch geval en is daarom in Sectie 5.2.3 besproken.

Een *zevende* regel behandelt de opeenvolging van de consonanten. Hierbij moet er rekening gehouden worden dat sommige combinaties niet gewenst zijn. De algemene regel die Zarlino geeft, zegt dat de ene consonant naar de andere consonant moet oplossen, die het dichtste in de buurt ligt. Als voorbeeld gaat een grote sext naar een octaaf, terwijl de kleine sext zich oplost naar een kwint. Algemener gezegd gaat een *groot* interval naar een interval dat groter is, net zoals een *klein* interval zich oplost naar een kleiner interval.

Tenslotte moet men er hier ook mee rekening houden, dat een stem zich niet te lang in de uiterste registers bevindt, zoals ook in de homofonie van toepassing is.

Listing 6.4: Afwisseling tussen perfecte en imperfecte consonanten

```
/*
  afwisselingsconstraint
      afwisseling_perfect_imperfect (Sopraan,Tenor) :-
          eerste interval is een imperfect interval ,
          de volgende een perfect interval .
      afwisseling_imperfect_perfect (Sopraan,Tenor) :-
          eerste interval is een perfect interval ,
          de volgende een imperfect of een perfect interval .
    ( lijst_van_integers ,
      lijst_van_integers ) (in,in)
 */

 afwisseling_perfect_imperfect ([],[]).
 afwisseling_perfect_imperfect ([SopraanNoot|RestSopraan],[TenorNoot|RestTenor])
:-
     imperfecte_consonant (SopraanNoot,TenorNoot),
      afwisseling_imperfect_perfect (RestSopraan,RestTenor).

 afwisseling_imperfect_perfect ([],[]).
 afwisseling_imperfect_perfect ([SopraanNoot|RestSopraan],[TenorNoot|RestTenor])
:-
     perfecte_consonant (SopraanNoot,TenorNoot),
      afwisseling_perfect_imperfect (RestSopraan,RestTenor).
 afwisseling_imperfect_perfect ([SopraanNoot|RestSopraan],[TenorNoot|RestTenor])
:-
     perfecte_consonant (SopraanNoot,TenorNoot),
      afwisseling_imperfect_perfect (RestSopraan,RestTenor).
```

Listing 6.5: De stemmen mogen niet te parallel lopen

```
  /*
    niet_te_parallel ( Intervallen ).
    Er mogen niet te veel parallellen inzitten
    ( lijst_van_integers ) (in)
   */

   niet_te_parallel ([Int1,Int2,Int3]) :-
    (#\ (( Int1 #= Int2 ) #/\ ( Int1 #= Int3 ) #/\ ( Int2 #= Int3 ))).
   niet_te_parallel ([Int1,Int2,Int3|Rest]) :-
    (#\ (( Int1 #= Int2 ) #/\ ( Int1 #= Int3 ) #/\ ( Int2 #= Int3 ))),
     niet_te_parallel ([Int2,Int3|Rest]).
```

6.2.4 Het toevoegen van de andere stemmen

Als we de bas en de alt willen toevoegen, terwijl we de tenor en de sopraan beschouwen is het belangrijk om de combinatie van intervallen te bekijken die de sopraan vormt ten opzichte van de tenor en anderzijds, de combinatie tussen de bas en de tenor en de combinatie van de bas en de alt. In tabel 6.1 staan alle combinaties opgesomd. Naar gelang het interval tussen de sopraan en tenor zijn er verschillende intervallen mogelijk tussen de bas en de tenor. In de laatste kolom worden de intervallen tussen de alt en de bas opgesomd, die dan nog mogelijk zijn na keuze van het interval tussen de bas en de tenor. Als de alt een priem of octaaf verschil mag zijn met de bas, dan mag de alt dit ook met de andere stemmen.

Sopraan en tenor	**Tenor en bas**	**Alt en Bas**
priem	terts	kwint, sext
	kwint	terts
	sext	terts
	octaaf	terts,kwint,sext
terts	terts	priem,octaaf
	sext	terts
	octaaf	kwint,sext
kwart	kwint	terts
kwint	octaaf	terts
	sext	priem,octaaf
sext	kwint	priem,octaaf
	terts	kwint
octaaf	terts	terts,kwint,sext
	kwint	terts
	octaaf	terts,kwint

Tabel 6.1: De mogelijke combinaties van intervallen voor vierstemmig contrapunt

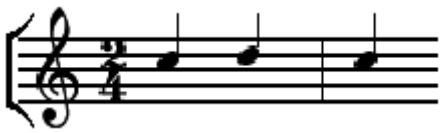
Hoofdstuk 7

De resultaten

Harmonie kan ik je leren,
maar voor contrapunt,
moet je een natuurlijke aanleg hebben.
Bernhard Zweers (1854-1924)

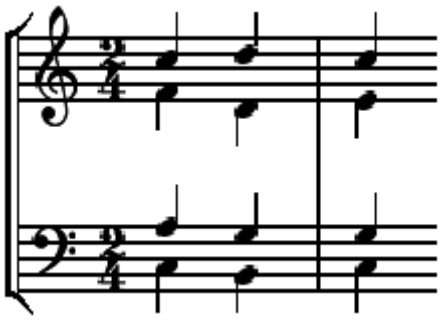
Een vergelijking tussen de methode van de harmonie en de methode van het contrapunt gaan we nu trekken aan de hand van de generatie van 2 voorbeelden. We laten telkens de module van de harmonie en de module van het contrapunt op onze voorbeelden, oplossingen genereren. Hiervan geven we de eerste oplossing weer die het programma gevonden heeft samen met het aantal oplossingen en de benodigde tijd.

Voor het eerste voorbeeld met maar drie noten is de nodige tijd te verwaarlozen voor de 2 methodes. De toonladder van de melodie is do groot. De harmonie slaagt erin om 12 oplossingen te genereren, waarvan de eerste te zien is in figuur 7.2. Bij contrapunt zien we 8 oplossingen, met als eerste oplossing figuur 7.3.

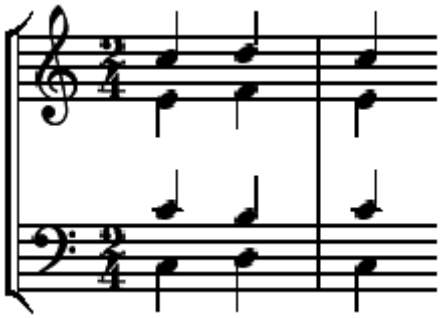


Figuur 7.1: Het voorbeeld met 3 noten

Als we alle oplossingen overlopen, merken we dat de oplossingen via harmonie totaal verschillen van de oplossingen via contrapunt. Als we de regels overlopen, kunnen we dit zelfs veralgemenen en beweren dat er in het programma nooit oplossingen zullen zijn die elkaar overlappen vanwege een restrictie die in de eenvoudige harmonieleer is bijgevoegd, in tegenstelling tot een andere regel in het contrapunt. Er wordt in de homofonie namelijk verwacht dat de akkoorden volledig voorkomen in de 4 stemmen, zodat je telkens 3 verschillende noten hebt in de 4 stemmen. Dit



Figuur 7.2: Het voorbeeld met 3 noten geharmoniseerd

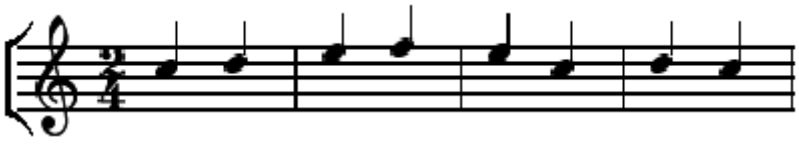


Figuur 7.3: Het voorbeeld met 3 noten via contrapunt

zou een volle klankkleur moeten opleveren. In de gewone harmonie mag men van deze regel echter afwijken. Als we gaan kijken naar de laatste noten van de voorbeelden, dan wordt er bij contrapunt verwacht, dat de tenor eindigt op een octaaf verschil van de sopraan. De laatste noot van de bas moet op zijn beurt ook een priem of een octaaf met de laatste noot van de tenor zijn. Dit zorgt ervoor dat alleen de alt een verschillende noot kan zijn ten opzichte van de andere. Men heeft in het laatste akkoord dan ook slechts 2 verschillende noten, wat nooit naar een volledig akkoord volgens de eenvoudige harmonieleer kan leiden.

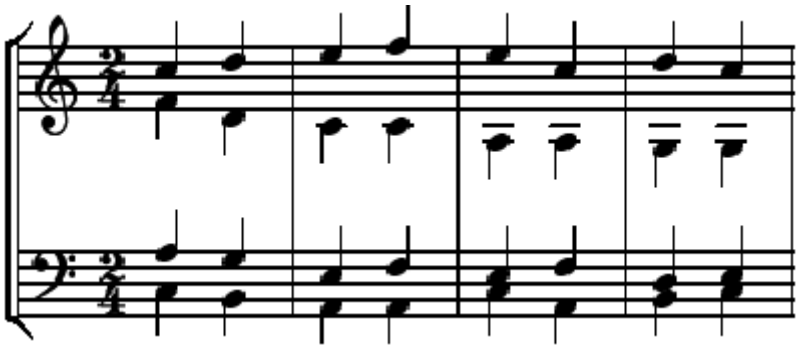
De regels van contrapunt zijn op dit korte voorbeeld ook heel streng. Als we de oplossingen van dichtbij bekijken, merken we dat de tenor en de bas vastliggen. Enkel de alt kan variaties geven. Dit komt deels door de bepaling dat de laatste noot van de tenor en de bas vast staan en deels doordat het begin ook een perfecte consonant moet zijn. Als laatste heeft hier ook de beperking dat de noten zich naar de dichtsbijzijnde consonant zich moeten begeven en niet te veel mogen springen een grote invloed. Bij de harmonie is er meer vrijheid zodat elke stem meerdere mogelijkheden heeft.

Als we vóór het vorige voorbeeld 5 noten bijplaatsen en dezelfde toonladder behouden, dan komen er een aantal mogelijke oplossingen bij, die ervoor zorgen dat er een verschil is in het aantal oplossingen en in de benodigde tijd.

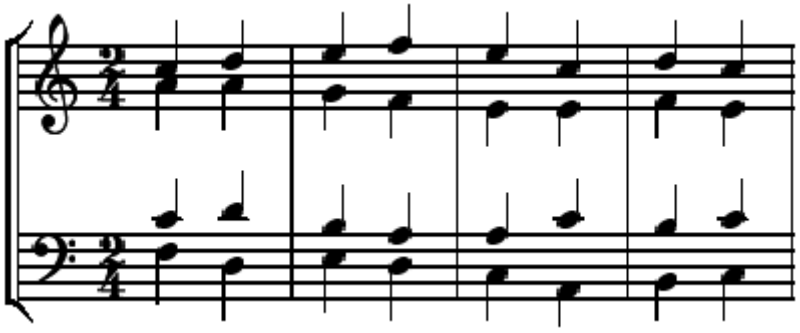


Figuur 7.4: Het voorbeeld met 8 noten

Via de harmonie worden er 1139 mogelijkheden goed bevonden in 20 seconden. Via contrapunt vinden we 28 oplossingen in 16 seconden.



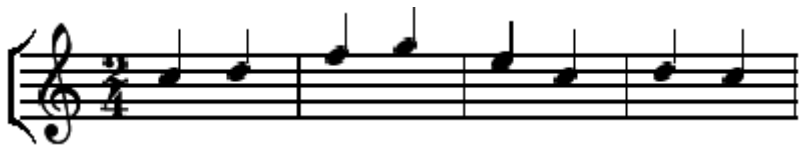
Figuur 7.5: Het voorbeeld met 8 noten geharmoniseerd



Figuur 7.6: Het voorbeeld met 8 noten via contrapunt

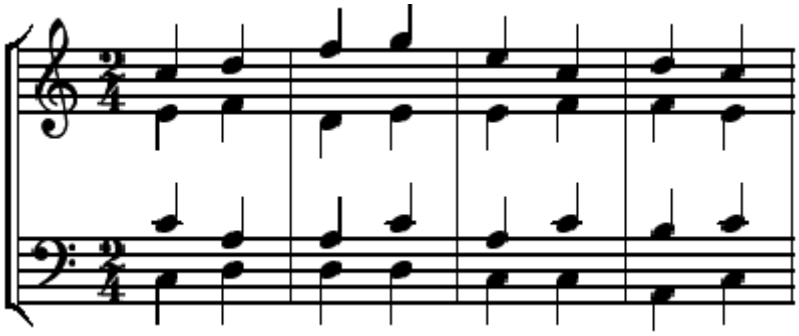
Hier zien we met 8 noten al een klein tijdsverschil en een groot verschil in mogelijkheden. Het verschil in tijd heeft te maken kunnen hebben met het zoeken naar akkoorden wat niet in CLP gebeurt. Een andere reden zou te maken kunnen hebben met het aantal oplossingen die dat weggeknipt worden uit het domein omwille van de strengere regels van contrapunt t.o.v. harmonie. Deze laatste reden geeft alvast een verklaring aan het verschil in oplossingen.

Als we nu de sequentie wat aanpassen aan de volgende melodie krijgen we echter totaal andere resultaten.



Figuur 7.7: Het tweede voorbeeld met 8 noten

In 50 seconden weet het programma te vertellen dat er via contrapunt 112 mogelijkheden zijn.



Figuur 7.8: Het tweede voorbeeld met 8 noten via contrapunt

Via harmonie gaat het deze keer vlotter dan bij contrapunt. In 20 seconden vertelt het programma dat er geen oplossingen gevonden zijn. Kijkend naar de versie van contrapunt zou dit kunnen doordat er bij de homofonie een beperking bijzit i.v.m. de variatie van stemmen en van akkoorden. Het programma laat geen oplossingen toe waar een stem 3 maal achter elkaar dezelfde noot moet zingen en ook niet dat 3 maal hetzelfde akkoord gebruikt wordt. Dit wil zeker niet zeggen dat de harmonie dit probleem niet zou kunnen oplossen. De eenvoudige harmonieleer samen met de variatieregel zorgen er echter voor dat er soms te streng wordt opgetreden.

Hoofdstuk 8

Besluit

8.1 Samenvatting

In deze tekst en in het programma zijn we op zoek gegaan naar regels die toepasbaar zijn om een éénstemmige compositie te verrijken tot een vierstemmige compositie. We steunden hier op twee methodes, namelijk die van het contrapunt volgens Zarlino (zonder dissonanten) en via een eenvoudige harmonieleer, die een afleiding en een beperking is van de klassieke harmonieleer. Hierbij is rekening gehouden met het proberen indijken van het aantal oplossingen.

Deze regels zijn in CLP geïmplementeerd. Het uiteindelijke programma is goed bruikbaar om voor stukken uit een compositie, binnen een redelijke tijd oplossingen te genereren. Met een redelijke tijd wordt er minder dan één minuut bedoeld. Enkel het zoeken van akkoorden in de harmonieleer wordt met een genereer- en testmethode gedaan, omdat sommige akkoorden een voorkeur genieten ten opzichte van de andere.

8.2 Resultaat

Als we het programma oplossingen laten genereren voor een sequentie van 8 noten, dan merken we dat er een groot verschil is in aanpak via de harmonieleer en via de leer van contrapunt. Dit vertaalt zich in een tijdsverschil in het zoeken van oplossingen en in het aantal oplossingen. Bij de ene melodie vindt de eenvoudige harmonieleer zelfs geen oplossingen. Vanwege bepaalde beperkingen die in de eenvoudige harmonieleer worden opgelegd ten opzichte van andere beperkingen in de volledige harmonieleer kan er geen volledig beeld gemaakt worden van het verschil tussen de harmonieleer en het contrapunt. Er wordt wel verwacht aan de hand van de huidige beperkingen, dat bij uitbreiding naar de volledige harmonieleer er veel meer oplossingen mogelijk zullen zijn.

8.3 Toekomstmuziek

Het zou te ambitieus zijn om de harmonie en de polyfonie volledig te implementeren. Daarom heb ik er voor gekozen om een eenvoudige harmonieleer te gebruiken. Als vervolg op dit werk, ligt het dan ook voor de hand dat men dit gaat uitbreiden met de andere regels. Omdat dit waarschijnlijk meer oplossingen zal geven zal het zwaartepunt liggen op het zoeken naar de beste oplossing.

Ook kan er zoals in hoofdstuk 5.3 beschreven is de module van akkoorden herschreven worden, zodat het volledig in CLP werkt.

Voor de polyfonie is er gekozen om geen dissonanten te gebruiken. Dit soort contrapunt noemt men het simpele contrapunt. De uitbreiding ligt voor de hand om deze dissonanten toe te voegen aan de regels. Ook kan er dan gevarieerd worden met de ritmes, zodat ook deze parameter van een noot ten volle gebruikt wordt.

Buiten deze uitbreiding van de regels kan men ook andere regels gaan toepassen zoals die beschreven zijn door Ernest W. Mulder voor polyfonie[5] of harmonie[2]. Ook de regels van Palestrina kunnen voor de polyfonie geïmplementeerd worden.

Buiten het verder bouwen om de leer van de harmonie en het contrapunt kunnen we ook andere vormen van muziek gaan implementeren. Om binnen de polyfonie te blijven, is het voor de hand liggend dat canons en fuga's in aanmerking komen. Daarbuiten zijn er de verschillende muziekstijlen zoals Blues en Rock.

Ook andere vormen van programmeertalen kunnen hiervoor gebruikt worden. Op deze manier kan er een vergelijkende studie gemaakt worden i.v.m. de snelheid en de aanpasbaarheid van de programma's. Hierbij kan men gebruik maken van CP (*constraint programming*), wat een veralgemening is van CLP. Hetzelfde paradigma wordt gebruikt, maar het beperkt zich niet alleen tot de logische programmeertalen, maar kan ook toegepast worden in Java of een andere taal, zodat er nog altijd kan gewerkt worden met het concept van een verzameling regels.

Om het interne formaat leesbaar te maken naar de dirigent of muzikant moet dit kunnen omgezet worden naar een leesbaar formaat. Na wat onderzoek zou dit het beste kunnen door een omzetting naar het *Lilypond*-formaat. Hiervoor is er een programma vrij beschikbaar die de eenvoudige tekstuele voorstelling van de noten kan omzetten naar notenbalken in het pdf-formaat en naar geluidsbestanden in het midi-formaat.

Als laatste kan het de dirigent nog gemakkelijker gemaakt worden door een grafische interface te maken, waar de noten naar toe kunnen versleept worden of waar de invoer direct zichtbaar op het scherm komt. Door een druk op de knop kan het achterliggende programma zijn werk doen, zonder dat de dirigent weet dat dit in Prolog gebeurt.

Bibliografie

[1] Neil V. Hawes. `http://ourworld.compuserve.com/homepages/Neil_Hawes/theory.htm`.

[2] Ernest W. Mulder. *Harmonie. 2: De practische harmonieleer*. De Haan Utrecht, 1952.

[3] T. K. B. productions. `http://www.tkb-productions.nl/overmuziek/woordenboek/ltmz.html`.

[4] Musys Software. `http://www.musicad.nl/theorie.htm`.

[5] Ernest W. Mulder. *Polyphonie*. De Haan Utrecht, 1955.

[6] Gioseffo Zarlino. *The art of counterpoint*. Da Capo New York (N.Y.), 1983.

[7] Jean-Philippe Rameau and Deborah Hayes. *Rameau's theory of harmonic generation: an annotated translation and commentary of gnration harmonique by Jean-Philippe Rameau*. University microfilms international Ann Arbor (Mich.), 1979.

[8] Gioseffo Zarlino. *Le istitutioni harmoniche*. Broude New York (N.Y.), 1965.

[9] Francesca Rossi. Constraint logic programming.

[10] Vitor Nogueira, Salvador Abreu, and Gabriel David. Towards temporal reasoning on isco. In Julio Mariño Carballo Juan José Moreno-Navarro, editor, *Proceedings of the Joint Conference on Declarative Programming APPIA-GULPE-PRODE*, pages 311–324, Madrid, Spain, September 2002.

[11] Joxan Jaffar and Michael J. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19/20:503–581, 1994.

[12] Kim Marriott and Peter J. Stuckey. *Programming with Constraints: An Introduction*. MIT Press, 1998.

[13] Hana Rudová. *Constraint Satisfaction with Preferences*. PhD thesis, Faculty of Informatics, Masaryk University, 2001.

[14] R. Bartak. Constraint programming: In pursuit of the holy grail, 1999.

[15] Richard Alexander Vriens. `http://members.lycos.nl/decodexdermuziek`.

[16] Hana Rudová. Soft CLP(FD). In Susan Haller and Ingrid Russell, editors, *Proceedings of the 16th International Florida Artificial Intelligence Symposium, FLAIRS-03*. AAAI Press, 2003.

[17] Plato, Benjamin Jowett, and Fritz Kredel. *The republic*. Heritage press New York (N.Y.), 1944.