Et system til studieplanlægning

Peter Holm Østergaard, s103459
Nicolai Vindahl Polack, s103470

Kongens Lyngby 2013
IMM-B.Sc.-2013-13

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk IMM-B.Sc.-2013-13

Abstract

The goal of this thesis is to create a study planning system, which students of DTU can use to plan their curriculum. We create this system by reducing the problem of study planning, to a problem in Satisfiability Modulo Theory, where each study plan requirement is reduced to a constraint in the SMT-problem. We choose to represent the reduced problem using an integer variable corresponding to each possible course in the curriculum.
We implement this system in F# using the SMT-solver Z3. To uptain the necessary information about the courses and rules at DTU, we use the DTU resources "kursusbasen" and "studiehåndbogen" respectively.

Indledning

Opgavebeskrivelse
Formålet med denne opgave er, at udarbejde et system til studieplanlægning for studerende på DTU. Dette system skal på baggrund af information om DTUs kurser og uddannelser, kunne generere studieplaner for både bachelor- og kandidatuddannelser. Brugeren af systemet skal kunne stille en række krav til studieplanen, som systemet hvis muligt skal opfylde. Vi ønsker at implementere vores system i programmeringssproget F#. Vi vil i opgaven lægge vægt på at designe og implementere vores system, fremfor at udvikle et program med en brugervenlig grafisk brugergrænseflade.

Metode
Vi ønsker at løse opgaven ud fra følgende fremgangsmåde:

1) Specificere hvilke krav brugeren skal kunne stille til studieplanen.

2) Specificere hvilke formelle krav der er til DTUs uddannelser, og som studieplanen dermed skal kunne opfylde.

3) Reducere disse krav til et problem i Satisfiability Modulo Theory.

4) Lave en database i F# med alle udbudte kurser på DTU, og deres tilhørende information.

5) Implementere vores system til generering af studieplaner i F#, ved hjælp af SMT-solveren Z3.

Vi vil i denne forbindelse benytte DTUs studiehåndbog som vores kilde til de formelle krav til uddannelserne, og DTUs kursusbase som vores kilde til alt information om de udbudte kurser på DTU.

Rapportens disposition
Kapitel 1: *Introduktion til studieplaner*
I dette kapitel vil vi, med udgangspunkt i en konkret studieplan, give en generel introduktion til studieplaner. Vi vil i denne sammenhæng også definere relevante nøgleord og begreber, som vi vil benytte i løbet af resten af rapporten.

Kapitel 2: *Kravspecifikation*
Vi vil i dette kapitel definere hvad der udgør en studieplan i vores system, og hvilke krav denne skal opfylde. Denne definition og disse krav vil vi så bruge til at formulere en kravspecifikation for vores program.

Kapitel 3: *Formalisering til SMT-problem*
Vi vil her give en kortfattet beskrivelse af Satisfiability Modulo Theory. Dernæst vil vi beskrive hvordan vores krav til systemet kan reduceres til et SMT-problem, og hvordan vores konkrete reduktion ser ud.

Kapitel 4: *Design*
I dette kapitel vil vi beskrive hvordan vi har designet vores system, og redegøre for de designmæssige valg vi har truffet i forbindelse med arbejdet. Vi vil desuden give et overblik over vores valg af typer og funktioner i implementeringen af systemet.

Kapitel 5: *Z3*
Vi vil i dette kapitel give en introduktion til Microsofts SMT-solver Z3. Vi vil kort beskrive teorien bag denne solver, samt give et eksempel hvordan den kan bruges i praksis.

Kapitel 6: *Implementering*
Her vil vi først give et generelt overblik over vores implementering, og hvordan programmet fungerer. Dernæst vil vi gennemgå en række udvalgte funktioner, og redegøre for hvordan vi har implementeret dem.

Kapitel 7: *Test*
I dette kapitel vil vi teste at alle krav i vores kravspecifikation er opfyldt, samt teste for fejl i en række funktioner i vores progam.

Kapitel 8: *Anvendelse*
Vi her demonstrere hvordan en studerende med en given række krav, kan bruge vores program til at generere nyttige studieplaner.
Kapitel 9: *Videreudvikling*

Til sidst vil vi beskrive hvilke muligheder der er for at arbejde videre på vores projekt, og hvordan man kunne løse nogle af de problemer der er med systemet.

Læserens faglige kendskab
Vi antager at læserens faglige kendskab er tilsvarende det for en studerende på DTU-uddannelsen Bachelor i Softwareteknologi. Vi antager dermed at læseren på forhånd er bekendt med programmeringssproget F#, og har kendskab til logik og mængdelære. Mere avancerede emner, som f.eks. Satisfiability Modulo Theory og Z3, vil blive forklaret i rapporten.

Indholdsfortegnelse

Kapitel 1

Introduktion til studieplaner

Vi vil her starte med at give en introduktion til studieplaner ved vise et eksempel på en konkret studieplan, samt definere en række nøglebegreber der hører til DTUs kurser og uddannelser.

1.1 Studieplan

Her ses et eksempel på en studieplan for en bachelor studerende der læser på studieretningen software-teknologi:

<table>
<tr><td>Semester</td><td colspan="5">13-ugers periode forår/efterår
(25 ECTS)</td><td>3-ugers periode jan/jun
(5 ECTS)</td></tr>
<tr><td>1.</td><td colspan="2" rowspan="2">Matematik 1</td><td>Diskret matematik</td><td>Indledende programmering</td><td colspan="2">Introduktion til softwareteknologi</td></tr>
<tr><td>2.</td><td>Algoritmer og datastrukturer 1</td><td>Software engineering 1</td><td>Grafteori</td><td>Statistik</td></tr>
<tr><td>3.</td><td rowspan="2">Fysik 1</td><td>Grundlæggende kemi</td><td colspan="2">Indlejrede systemer</td><td>Algoritmer og datastrukturer 2</td><td>Indledende programmering med Matlab</td></tr>
<tr><td>4.</td><td colspan="2">Datalogisk modellering</td><td>Introduktion til kunstig intelligens</td><td colspan="2">Fagprojekt</td></tr>
<tr><td>5.</td><td>Logiske Systemer og Logikprogrammering</td><td>Funktions-programmering</td><td>Parallel-programmering</td><td>Windows Programming using C# and .Net</td><td>Programmering i C++</td><td>Anvendt Funktions-programmering</td></tr>
<tr><td>6.</td><td>Videnskabsteori</td><td>Data-logik</td><td>Formelle aspekter af software engineering</td><td colspan="3">Bachelorprojekt</td></tr>
</table>

Figur 1.1

Som det fremgår af eksemplet, består en studieplan af en række kurser, der hver især ligger på et bestemt semester. Hvert kursus ligger desuden enten i en 13-ugers periode eller en 3-ugers periode. For eksempel ligger kurset *Indledende Programmering* i 13-ugers perioden i første semester. Denne studieplan strækker sig over 6 semestre, og har kurser for 30 ECTS point på hvert semester. Farverne i studieplanen angiver hvilken kursustype, kurserne har. Eksempelvis er *Matematik 1* et Naturvidenskabeligt grundfag (jf. definitionen af kursustype). Der er en række informationer omkring kurserne der ikke fremgår af eksemplet, da de ikke er nødvendige at kende for en færdig studieplan. Dette er informationer som f.eks. antal ECTS point for et kursus, kursets skemaplacering, kursets forudsætninger, og mange flere.
Vi vil nu definere alle de begreber der er relevante for en studieplan, med udgangspunkt i eksemplet vist ovenfor.

1.2 Begrebsdefinitioner

Alle definitioner og eksempler på krav i dette afsnit er lavet med udgangspunkt i DTUs studiehåndbog. Studiehåndbogen er en samling af alle informationer og krav til DTUs uddannelser.

1.2.1 Semester

Enhver studieplan er opdelt i en række semestre. Et semester består af en 13-ugers periode og en 3-ugers periode i enten efteråret eller foråret. Et semester indeholder typisk kurser for 30 ECTS point, men kan indeholde både mere eller mindre.

1.2.2 Skemaplacering

DTU bruger en række skemaplaceringer til at beskrive hvilket tidspunkt et kursus foregår. Nedenfor ses et skema for en 13-ugers periode, hvor det fremgår hvordan skemaet er opdelt i forhold til hver skemaplacering. Da en 13-ugers periode enten kan ligge i efteråret eller foråret, skriver man henholdsvis E eller F foran skemaplaceringen, således at mandag formiddag enten hedder E1A eller F1A.

	Mandag	Tirsdag	Onsdag	Torsdag	Fredag
8-12	1A	3A	5A	2B	4B
12-13	Pause	Pause	Pause	Pause	Pause
13-17	2A	4A	5B	1B	3B

Figur 1.2

Udover de 10 primære skemaplaceringer vist i skemaet, er der fem sammensatte skemaplaceringer, der hver især består af to af de primære skemaplaceringer. For eksempel vil skemaplaceringen, der dækker over E1A og E1B, hedde E1. Hvis et kursus ligger på en af de fem sammensatte placeringer, kan et andet kursus ikke ligge på nogle af de primære placeringer, som den sammensatte dækker over. Det samme gælder den anden vej. For eksempel kan et kursus med den sammensatte skemaplacering E1 ikke vælges, hvis de primære skemaplaceringer E1A eller E1B er optaget.
En 3-ugers periode har én skemaplacering, der dækker over hele 3-ugers perioden. Man kan således kun have ét kursus i denne periode.

1.2.3 Kursus

Hvert kursus består af følgende informationer:

- Et kursusnummer
- Et navn
- En eller flere skemaplaceringer
- Et antal ECTS point
- Et kursusniveau (jf. definition af kursusniveau)
- En kursustype (jf. definition af kursustype)
- Eventuelle kurser der pointspærres for (jf. definition af pointspærring)
- Eventuelle obligatoriske forudsætninger (jf. definition af obligatoriske forudsætninger)
- Eventuelle fagområder (jf. definition af fagområde)

Hvis man tager kurset *Indledende Programmering* fra eksemplet på en studieplan, vil det have følgende informationer:
Kursusnummeret 02101, navnet Indledende Programmering, skemakoden E3A, 5 ECTS point, det er et grundlæggende kursus, har kursustypen Projekter og almene fag (Softwareteknologi) og pointspærrer for kurserne 02100, 02102, 02115, 02199 og 02312.

1.2.4 Kursusniveau

Et kursus kan enten være et grundlæggende-, videregående- eller ph.d-kursus, afhængigt af sværhedsgraden af kursets pensum. Der findes forskellige krav til DTUs uddannelser med hensyn til kursusniveau, f.eks. at man ikke må tage ph.d-kurser på en bacheloruddannelse.

1.2.5 Kursustype

Der findes forskellige kursustyper for bachelor- og kandidatuddannelser. En kursustype gælder altid kun i forhold til en bestemt studieretning.
For bacheloruddannelser findes der kursustyperne:

- Naturvidenskabelige grundfag
- Teknologiske linjefag
- Projekter og almene fag
- Valgfri fag

Disse kursustyper er vist med henholdsvis rød, blå, gul og grå i eksemplet på en studieplan.
For kandidatuddannelser findes der kursustyperne:

- Generelle Retningskompetencer
- Teknologisk Specialisering
- Valgfri fag

Med mindre andet er angivet, antages det at alle kurser har kursustypen valgfri fag.

DTU har forskellige krav til uddannelserne omkring kursustyper, f.eks. at en bacheloruddannelse skal bestå af kurser for mindst 45 point med kursustypen Naturvidenskabelige grundfag, Teknologiske linjefag og Projekter og almene fag.

1.2.6 Pointspærring

At et kursus pointspærrer for et andet kursus vil sige, at man ikke kan få point for begge kurser samtidig. Som et eksempel pointspærrer det statistikkursus, der ligger i 13-ugers perioden, for det statistikkursus, der ligger i 3-ugers perioden. Hvis man derfor tager begge kurser, får man kun point for ét af dem.

1.2.7 Obligatoriske forudsætninger

Et kursus' obligatoriske forudsætninger fremgår som en række af andre kurser, der skal være bestået før selve kurset kan tages. Det vil i en kursusplan betyde, at forudsætningskurserne skal vælges i en tidligere periode end selve kurset. For eksempel har kurset *Data-logik* kurset *Logiske Systemer og Logikprogrammering* som obligatorisk forudsætning. Det vil sige at *Logiske Systemer og Logikprogrammering* i kursusplanen skal vælges i en tidligere periode end *Data-logik*.

1.2.8 Fagområde

Hvert kursus kan siges at høre under et eller flere fagområder. Et eksempel kunne være *Indledende Programmering*, der bl.a. kunne hører under fagområderne *Java*, *Programmering*, *objekt-orienteret programmering* og *imperativ programmering*.

Kapitel 2

Kravspecifikation

Vi vil i dette kapitel først overveje de behov vi forsøger at imødekomme i vores projekt, ved at beskrive de forskellige overvejelser vi har gjort os, og hvilke funktioner systemet bør indeholde. På baggrund af dette, vil vi lave en definition af en studieplan, samt lave en konkret kravspecifikation der præciserer hvilke krav vores system skal opfylde.

2.1 Overvejelser og definition af studieplan

Vores målgruppe for dette studieplanlægningsprogram er civilingeniør-studerende på DTU.
Programmet skal kunne bruges af alle civilingeniør-studerende uanset om de ønsker at planlægge et bachelor-studie elleret kandidat-studie. Det skal også kunne bruges af studerende der befinder sig midt i et studieforløb, og ønsker at planlægge enten resten af forløbet, eller blot et enkelt semester.
Det skal være muligt for den studerende at få opfyldt alle de almindelige krav der kan opstå i forbindelse med planlægningen af et studieforløb. Eksempelvis kan den studerende have et studiejob hver mandag, og kan dermed ikke have undervisning den dag.
Vi har valgt at fokusere på studieplanlægning af DTUs uddannelser, da vi har

mere genskab til disse end til uddannelser på andre universiteter. Derudover har vi valgt at afgrænse projektet ved at se bort fra diplomingeniør-uddannelser, da disse har helt andre gældende krav i studiehåndbogen end civilingeniør-uddannelserne har.

Programmet skal først og fremmest kunne generere en gyldig studieplan, dvs. en studieplan for et helt uddannelsesforløb der tager højde for alle kravene fra studiehåndbogen. Derudover bør man ikke kunne vælge et kursus mere end én gang, da man i en studieplan antager at bestå alle valgte kurser. For at gøre studieplanen mere realistisk bør man heller ikke kunne vælge mere end ét kursus på samme skemaplacering på samme semester. På baggrund af disse krav, samt de formelle krav DTU stiller, har vi lavet følgende definition på en gyldig studieplan:

Definition på en gyldig studieplan
En gyldig studieplan er en række kurser, hvor det for hvert kursus er angivet hvilket semester kurset skal tages på. En studieplan kan enten være for en bachelor- eller en kandidatuddannelse.
En gyldig studieplan er defineret ud fra følgende punkter:

S1: En studieplan for en bachelor- eller kandidatuddannelse udgør hhv. 180 eller 120 ECTS point.

S2: En studieplan for en bachelor kan maksimalt bestå af 12 semestre, og en kandidat 8 semestre.

S3: En studieplan skal være tilknyttet en studieretning.

S4: Hver studieretning har en række obligatoriske kurser, som skal følges, herunder fagprojekt, bachelorprojekt og kandidatspeciale.

S5: En studieplan for en kandidatuddannelse kan maksimalt have for 10 ECTS point grundlæggende kurser.

S6: En studieplan for en bacheloruddannelse kan kun bestå af grundlæggende og videregående kurser.

S7: En bacheloruddannelse skal indeholde kurser svarende til mindst 45 ECTS point fra hver af kursustyperne:

- Teknologiske Linjefag, Naturvidenskabelige grundfag og Projekter og almene fag.

S8: Hver studieretning på kandidatuddannelsen har forskellige krav til kursustyperne som skal opfyldes. Eksempelvis skal der på retningen Informationsteknologi vælges kurser for mindst 30 ECTS point fra hver af kursustyperne:

– Generelle Retningskompetencer og Teknologisk Specialisering.

S9: Hvert kursus i studieplanen kan kun tages én gang.

S10: Der må ikke ligge flere kurser på samme skemaplacering i det samme semester.

S11: Kurser der bliver pointspærret for, må ikke vælges.

S12: Et kursus kan kun tages, hvis de kurser der er angivet som obligatoriske forudsætninger, er taget i en tidligere periode.

S13: Bachelorprojekt og kandidatspeciale må ikke vælges før der er valgt kurser for hhv. 90 og 55 ECTS point.

Da de færreste studerende holder sig til én studieplan gennem hele studiet, bør systemet derfor også kunne generere en partiel studieplan, dvs. en studieplan der kun delvist udgør en gyldig studieplan, og som ikke nødvendigvis opfylder alle DTUs krav.

Definition på en partiel studieplan
En partiel studieplan er defineret som følgende:

S14: En partiel studieplan er en studieplan, der skal kunne *udvides* til en gyldig studieplan.

Med udvides menes der, at yderligere kurser bliver tilføjet til studieplanen. Dermed vil punkterne *S2*, *S3*, *S5*, *S6*, *S9*, *S10*, *S11*, *S12* og *S13* også altid gælde for en partiel studieplan, samtidig med at den studerende altid kan udbygge studieplanen til også at opfylde punkterne *S1*, *S4*, *S7* og *S8*.
Et eksempel på en partiel studieplan kunne være første og andet semester i eksemplet på en studieplan fra forrige kapitel, hvor hele studieplanen svarer til en gyldig studieplan.

For at sådanne studieplaner kan blive genereret, skal systemet kende DTUs kurser og deres relevante oplysninger. Det vil derfor være oplagt at give systemet adgang til DTUs kursusdatabase, hvor alle disse informationer findes.
Systemet skal også bruge en række informationer fra brugeren. Her er det oplagt at skelne mellem de nødvendige informationer som programmet **skal** kende, og de informationer der **kan** være nødvendige for at generere en studieplan.
Det virker rimeligt at antage, at man på forhånd ved hvor mange semestre man ønsker at planlægge, og hvilken studieretning man ønsker at planlægge for. Uden disse informationer er det ikke muligt at lave en meningsfuld studieplan, da studieplanen ellers kan have en vilkårlig længde, og indeholde fuldstændigt

irrelevante kurser. Det giver derfor mening at disse to informationer **skal** angives til programmet.
Hvis man allerede er i gang med et studieforløb, og ønsker at planlægge resten, bør systemet kende til de kurser man har bestået tidligere, således systemet kan tage højde for kursusforudsætninger og studieretningskrav. Derved at det også muligt at tage højde for meritoverførsel, da brugeren dermed kan angive dette som tidligere beståede kurser. Ligeledes, hvis man ønsker at tage et semester i udlandet, skal det være muligt at oplyse dette i systemet. Dermed ved systemet at der ikke skal vælges kurser på det ønskede semester.
Vi ønsker at systemet skal kunne opfylde alle de almindelige krav en studerende kan have i forbindelse med studieplanlægning. Det skal derfor være muligt for brugeren at angive disse krav til systemet. Eksempler på sådanne krav kan være studiemæssige krav, som f.eks. hvilke kurser der skal være en del af studieplanen, hvilke skemaplaceringer der skal være fri, og om kurser skal vælges i de forskellige 3-ugers perioder. Det kan også være mere praktiske krav, som f.eks. hvilken årstid studieplanen skal starte på, hvor mange ECTS point hvert semester skal indeholde og hvorvidt studieplanen skal være gyldig eller partiel.

For at systemet giver mening at bruge, skal det være muligt at se den studieplan der bliver genereret. Der er flere muligheder for at vise en studieplan; den optimale måde for brugeren ville være ved hjælp af en grafisk brugerflade, der kunne repræsentere studieplanen i samme stil som i eksemplet fra det tidligere kapitel. Dette vil dog være en forholdsvis tidskrævende løsning, hvor den eneste gevinst er et mere visuelt behageligt system. Da vi hellere fokusere vores ressourcer på den bagvedliggende kode, har vi i stedet valgt at repræsentere en studieplan visuelt med en simpel tekstfil, hvor alle de valgte kurser er angivet med skemaplaceringer og semester.

Da studerende ofte udarbejder deres studieplan gennem flere iterationer, bør det være muligt at gemme genererede studieplaner. Ligeledes bør det være muligt for brugeren at kunne se sine krav efter at have genereret en studieplan, og således nemt ændre disse krav og generere en ny studieplan.

2.2 Kravspecifikation

Programmet skal have adgang til DTUs kursusdatabase, bestående af samtlige udbudte kurser på DTU. Ud fra denne database skal programmet for hvert kursus have adgang til de informationer der er angivet i kursus definitionen.
Programmet skal have en række input fra brugeren, som studieplanen skal genereres på baggrund af. Vi betragter her tre forskellige former for input:

Obligatoriske informationer:

- De informationer brugeren **skal** angive for at det er muligt at generere en studieplan.

Valgfrie informationer:

- De informationer brugeren **kan** angive, og som systemet skal tage højde for.

Valgfrie krav:

- De krav brugeren **kan** stille til den genererede studieplan.

Herunder følger for hver input-type, hvad programmet skal give brugeren mulighed for at angive. For de valgfrie informationer og krav er der angivet en standard input-værdi, svarende til at brugeren undlader at benytte det givne input:

Obligatoriske informationer:

OI1: Uddannelsens længde i semestre.
Eksempel: 6 semestre.

OI2: Studieretning for uddannelsen.
Eksempel: Softwareteknologi.

Valgfrie informationer:

VI1: Planlagt udlandsophold på et givent semester, for et givent antal ECTS point.
Eksempel: Udlandsophold på 3. semester for 30 ECTS point.

  – Hvis ikke angivet antages ingen udlandsophold.

VI2: Tidligere bestående kurser.
Eksempel: Matematik 1, Fysik 1 og Grundlæggende kemi.

  – Hvis ikke angivet antages ingen tidligere bestående kurser.

Valgfrie krav:

VK1: Om studieplanen skal være gyldig eller partiel.

- Hvis ikke angivet antages en partiel studieplan.

VK2: Om der skal vælges et kursus i 3-ugers perioden for de forskellige semestre.
Eksempel for studieplan på 4 semestre: Kurser skal vælges i 3-ugers perioden på 1. og 3. semester og må vælges på 2. og 4. semester.

- Hvis ikke angivet antages det at der skal vælges kurser i alle 3-ugers perioder.

VK3: Interval for antallet af ECTS point for hvert semester i studieplanen.
Eksempel for studieplan på 4 semestre: 25-30 ECTS point på 1. og 2. semester, 30-35 ECTS point på 3. og 4. semester.

- Hvis ikke angivet antages præcis 30 point på hvert semester.

VK4: Årstid for studieplanens start, enten efterår eller forår.

- Hvis ikke angivet antages efterårsstart.

VK5: Skemaplaceringer der **skal** være fri i angivne semestre.
Eksempel: E1A skal være fri på 1. semester og F2B skal være fri på 2. semester.

- Hvis ikke angivet antages ingen skemaplaceringer.

VK6: Skemaplaceringer der ønskes fri i angivne semestre.

- Hvis ikke angivet antages ingen skemaplaceringer.

VK7: Kurser der **skal** vælges i studieplanen, med mulighed for specifikt krav til semester.
Eksempel 1: Indledende Programmering skal vælges på første semester.
Eksempel 2: Indledende Programmering skal vælges på et vilkårligt semester.

- Hvis ikke angivet antages ingen kurser.

VK8: Kurser der ønskes valgt i studieplanen, med mulighed for specifikt krav til semester.

- Hvis ikke angivet antages ingen kurser.

VK9: Kurser der **ikke** må vælges i studieplanen, med mulighed for specifikt krav til semester.

- Hvis ikke angivet antages ingen kurser.

VK10: Krav til summen af ECTS point inden for bestemte fagområder.

- Hvis ikke angivet antages ingen specifikke krav.

Alle brugerens krav til programmet, undtagen to, skal tolkes som værende *hard constraints*. Dette vil sige at programmet skal opfylde samtlige krav, og må ikke give en løsning hvis dette ikke er muligt. De to undtagelser er *VK6* og *VK8*, som i stedet skal tolkes som værende *soft constraints*. Dette vil sige at programmet skal forsøge at opfylde disse krav, men at det ikke er nødvendigt at opfylde dem for at give en løsning.
Tilsammen udgør alle brugerens input den *kontekst*, som programmet bruger til at generere en studieplan.

På baggrund af kursusdatabasen og en given kontekst, skal programmet kunne opfylde følgende krav:

Programkrav:

K1: Programmet skal kunne indlæse en given kontekst fra en tekstfil.
Eksempel:

`Gyldig`	*Studieplanstype (Gyldig/Partiel)*
`Informationsteknologi`	*Studieretning*
`4`	*Studieplanslængde*
`25-35,30,30,25-35`	*Pointfordeling på semestre*
`Efterår`	*Årstid for studiestart (Efterår/Forår)*
`ja,nej,nej,ja`	*Fordeling af 3-ugers kursus på semestre*
`02157-1`	*Kurser der skal vælges*
`02257-0`	*Kurser der ønskes valgt*
`26000-0,Speciale-3`	*Kurser der ikke må vælges*
`E1A-1,E2A-1,F5-2,E4B-3`	*Skemaplaceringer der skal være fri på semesteret*
`E2A-1`	*Skemaplaceringer der ønskes fri på semesteret*
`30-4`	*Udlandsophold med angivet point på semesteret*
`01005,02101`	*Tidligere bestået kurser på bacheloren*
`02281`	*Tidligere bestået kurser på kandidaten*
`EOF`	*Signalerer at filen er slut*

Figur 2.1

K2: Programmet skal på baggrund af en given kontekst kunne generere en gyldig studieplan for både en bachelor- og kandidatuddannelse, der opfylder kravene *S1-S13* i definitionen af en gyldig studieplan.

K3: Programmet skal på baggrund af en given kontekst kunne generere en partiel studieplan, der opfylder kravet *S14* i definitionen af en partiel studieplan.

K4: Efter at have genereret en studieplan, skal studieplanen gemmes på tekstform, med valgte kurser angivet med skemaplacering og semester, samt se den tilhørende kontekst til studieplanen.
Eksempel:

```
Semester 1:
02165 - Udvikling af softwareprodukter - January
02203 - Design af digitale systemer - E2B
02239 - Datasikkerhed - E5B
02242 - Program analyse - E2A
02267 - Software-udvikling af Webtjenester - E1A
ECTS point: 30

Semester 2:
02224 - Tidstro systemer - F2B
02576 - Fysisk baseret rendering - F1A
30010 - Programmeringsprojekt - June
30530 - Geografiske informationssystemer - F4A
42490 - Teknologi, Økonomi, ledelse og organisation - F5
ECTS point: 30

Semester 3:
02156 - Logiske Systemer og Logikprogrammering - E4A
02217 - Design af aritmetiske processorer - E5B
02257 - Anvendt Funktionsprogrammering - January
02264 - Kravspecifikation - E3
02561 - Computer grafik - E5A
ECTS point: 30

Semester 4:
Mas30 - Speciale - June
Mas30 - Speciale - F
ECTS point: 30

-----------------------------------------------------
Denne studieplan er genereret ud fra følgende input:
Gyldig
Informationsteknologi
4
30,30,30,30
Efterår
ja,nej,nej,ja
Speciale-4









EOF
```

Figur 2.2

Kapitel 3

Reduktion af studieplans-generering til SMT-problem

Vores ønske om at kunne generere en studieplan på baggrund af en række krav, kan reduceres til et *Satisfiability Modulo Theory*-problem. Før vi kommer nærmere ind på hvordan vi vil gøre dette, vil vi kort beskrive den relevante teori, der er nødvendig at kende for at kunne lave en sådan reduktion.

3.1 Teori

3.1.1 Boolean Satisfiability problems

Indenfor datalogi støder man ofte på en type problemer, som man kalder "Boolean Satisfiability problems", eller *SAT*-problemer. Denne type problem drejer sig om at finde en fortolkning af et logisk udtryk, der gør udtrykket sandt. Man siger at en sådan fortolkning er en model for det givne udtryk. Et logisk udtryk der har en model siges at være *satisfiable*. Hvis alle fortolkninger af et udtryk er en model, siges udtrykket at være *valid*, og hvis der ikke findes en model er

udtrykket *unsatisfiable*.
Et eksempel på et sådant logisk udtryk kunne være:

$$p \vee \neg p$$

Dette udtryk har kun to forskellige fortolkninger: $p = true$ og $p = false$. Begge fortolkninger er en model for udtrykket, så udtrykket er både satisfiable og valid.
SAT er et *NP-complete* problem, hvilket betyder at der ikke findes en kendt algoritme der kan løse alle instanser af SAT-problemer i polynomiel tid.

3.1.2 Satisfiability Modulo Theories

Satisfiability Modulo Theories, eller *SMT*, kan bruges til at udtrykke problemer der enten er vanskelige eller umulige at udtrykke direkte som et SAT-problem. Dette gøres ved at betragte problemet i forhold til en bestemt kontekst af baggrundsteorier. Eksempler på sådanne teorier kunne være teori om aritmetik eller teori om forskellige datastrukturer, som f.eks. lister og tabeller. Det vil sige at symbolerne i et givent logisk udtryk ikke længere behøver at være logiske operatorer, som $\vee$ og $\wedge$, eller atomiske propositioner, som p og $\neg p$. I stedet kan man udtrykke et problem ved hjælp af f.eks. aritmetik, således at et udtryk også kan se sådan ud:

$$a + b > 3 \wedge a < 0 \wedge b > 0$$

En fortolkning af dette udtryk kunne være $a = -1$ og $b = 5$, hvilket er en model for udtrykket i konteksten af teori om aritmetik.
Man kan her betragte hvert led i udtrykket som en begrænsning af de fortolkninger der er modeller for udtrykket. Et sådant begrænsende led vi vil fremover kalde en *constraint*. Man kan altså udtrykke et SMT-problem som en række variabler samt en række constraints, der begrænser hvilke fortolkninger af variablerne der er sande.

3.2 Reduktion til SMT-problem

Helt overordnet set kan man reducere genereringen af en studieplan til et SMT-problem, ved at udtrykke de krav brugeren stiller og de formelle krav en studieplan

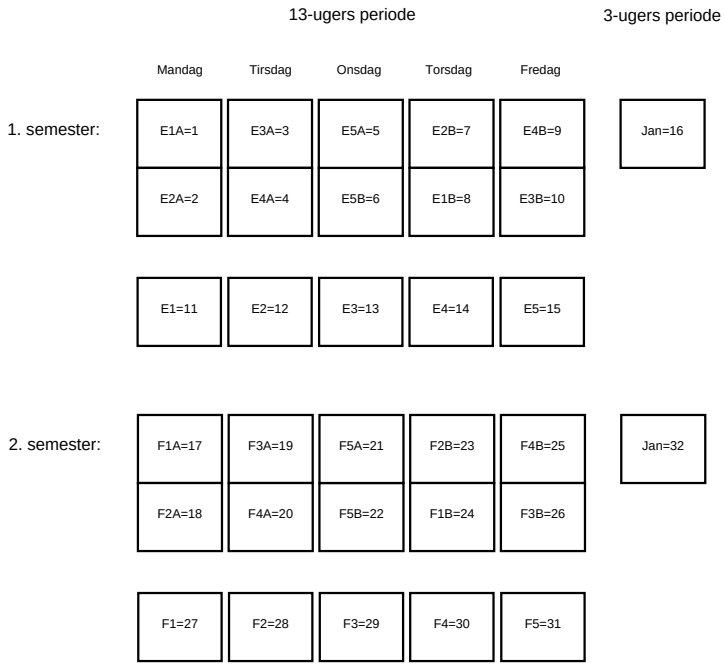
skal opfylde, som en række constraints der skal være opfyldt. Disse constraints kan dog udtrykkes på mange forskellige måder, afhængigt af hvilken type variablerne har, og hvad de dækker over.
Vi har valgt at bruge en repræsentation, hvor vi har en variabel svarende til hvert kursus, og variablens værdi er et heltal der svarer til det semester og skemaplacering kurset er taget på. Den variabel der svarer til kurset med kursusnummer 01005, kan dermed have denne form:

$$V_{01005} = 17$$

Hvis man så lader værdien 0 svare til at kurset ikke er valgt i studieplanen, vil man kunne repræsentere en partiel studieplan som følgende række variable:

$$V_{01005} = 17,\ V_{01007} = 0,\ V_{02021} = 12,\ \ldots,\ V_{02051} = 0$$

Det kræver dog at man ved hvilket semester og skemaplacering variablernes værdi svarer til. Vi har valgt at gøre det som vist i følgende figur:



Figur 3.1

Variablen $V_{01005} = 17$ betyder dermed at kurset med kursusnummer 01005 ligger på 2. semester med skemaplaceringen F1A. Semestret kan også udregnes ud fra værdien med formlen $((vaerdi - 1)/antal) + 1$, hvor *værdi* er variablens værdi, *antal* er antallet af skemaplaceringer på et semester, og '/' er heltalsdivision.
Ud fra denne repræsentation vil vi udtrykke alle kravene til studieplanen som en række constraints. Alle disse constraints kan laves som et udtryk i lineær heltals aritmetik. Vi har dog valgt også at benytte andre teorier til at udtrykke vores constraints, bl.a. teori om kvantoriserede udtryk og tabeller. Dette har vi valgt at gøre, da vores constraints ellers ville blive uoverskueligt store og ulæselige. Vi vil dog til sidst i dette kapitel vise hvordan en af vores constraints kan omskrives til et udtryk i lineær heltals aritmetik, med udgangspunkt i et konkret eksempel.

3.3 Constraints

Vi vil nu formulere de constraints, der skal bruges til at generere en korrekt studieplan. Til dette vil vi bruge følgende definitioner:

- Lad $C_i \in \mathbb{N}$ være en variabel, hvor i er et kursusnummer.
- Lad $C_i = s$ betegne at kurset med kursusnummer i vælges på skemakode s.
- Lad $C_i = 0$ betegne at kurset med kursusnummer i ikke vælges.
- Lad $E(i)$ være en funktion, der giver antallet af ECTS-point for kurset med kursusnummer i.
- Lad m være antallet af forskellige skemakoder på et semester.
- Lad n være antallet af semestre i studieplanen.
- Lad semester j bestå af kurserne med skemakoder i intervallet: $[m \cdot (j - 1) + 1; m \cdot j]$.
- Lad I være sættet af alle kursusnumre.
- Lad Iv være sættet af alle kursusnumre for alle valgte kurser.
- Lad TB være sættet af tidligere beståede kurser.

3.3.1 Constraints for en vilkårlig studieplan

Valgte kurser må ikke have samme skemakoder *(S10)*:

$$\forall i \in Iv : (\forall j \in (Iv \setminus \{i\}) : C_i \neq C_j)$$

Ingen point-spærrede kurser må vælges *(S11)*:
Lad $s(i)$ være en funktion, der giver sættet af alle de kursusnumre som kurset med kursusnummer i point-spærrer for.

$$\forall i \in Iv : (\forall j \in ((Iv \setminus \{i\}) \cup TB) : j \notin s(i))$$

Valgte kursers forudsætninger skal være opfyldt *(S12)*:
Lad $a(i)$ være en funktion, der giver et sæt af alle kombinationer af kursusnumre for kurser, der giver adgang til i.
Lad $ss(s)$ være en funktion, der giver den sidste skemakode i det forrige semester i forhold til skemakoden s.

$$\forall i \in Iv : (\exists j \in a(i) : (j \subseteq Iv \wedge (\forall k \in j : k \leq ss(C_i))) \vee j \in TB)$$

Ingen valgte kurser må ligge uden for det angivne antal semestre *(OI1)*:

$$\forall i \in Iv : 0 \leq C_i \leq n \cdot m$$

Der skal ikke vælges kurser på et semester, hvor der er valgt et udlandsophold *(VI1)*:
Lad UO være sættet af semestre hvor der er valgt et udlandsophold.

$$\forall i \in Iv : (\forall u \in UO : C_i \notin [m \cdot (u-1)+1; m \cdot u])$$

Tidligere beståede kurser skal ikke kunne vælges igen *(VI2)*:

$$\forall i \in Iv : i \notin TB$$

Et kursus skal være valgt i de givne 3-ugers perioder *(VK2)*:
Lad $(b_1, \ldots, b_n)$ være sandhedsværdier, hvor b_j betyder om et kursus **skal** vælges i 3-ugers perioden på semester j.

$$\forall b_j = true : (\exists C_i \in Iv : C_i = j \cdot m)$$

ECTS-point for alle semestre skal svare til den givne fordeling *(VK3)*:
Lad $(minE_1, \ldots, minE_n)$ være minimum-antallet af point per semester, hvor $minE_j$ er minimum-antallet af point på semester j.
Lad $(maxE_1, \ldots, maxE_n)$ være maksimum-antallet af point per semester, hvor $maxE_j$ er maksimum-antallet af point på semester j.

$$\forall j \in [1; n] : \sum_{for\ alle\ i} E(i) \geq minE_j \wedge \sum_{for\ alle\ i} E(i) \leq maxE_j,$$

$$\text{hvor } C_i \in [m \cdot (j-1) + 1; m \cdot j]$$

Ingen kurser må tages på de skemakoder der ikke må indeholde kurser *(VK5)*:
Lad US være sættet af skemakoder der ikke må benyttes.

$$\forall i \in Iv : C_i \notin US$$

Det maksimalt mulige antal uønskede skemakoder skal være fri *(VK6)*:
Lad L være sættet af alle løsninger, hvor hver løsning i sættet er et sæt bestående af kursusnumre for alle valgte kurser for denne løsning.
Lad US være sættet af skemakoder der ønskes fri.
Lad $aus(l, US)$ være en funktion, der giver antallet af opfyldte constraints fra US i en løsning l.

$$\forall l \in L : aus(l, US) \leq aus(Iv, US)$$

Kurser der *skal* tages på et bestemt semester, *skal* være en del af studieplanen på det givne semester *(VK7)*:
Lad $\emptyset K$ være sættet af alle par (k, s), hvor k er et kursusnummer for et kursus, der skal vælges, og s er det semester, det skal vælges på.
Lad $sem(sk)$ være en funktion, der giver det semester, som skemakode sk ligger på.

$$\forall (k, s) \in \emptyset K : (\forall i \in Iv : i = k \rightarrow sem(C_i) = s)$$

Kurser der *skal* tages på et vilkårligt semester, *skal* være en del af studieplanen *(VK7)*:
Lad $ØK$ være sættet af kursusnumre for alle kurser der **skal** tages.

$$\forall i \in ØK : i \in Iv$$

Det maksimalt mulige antal ønskede kurser skal være taget *(VK8)*:
Lad L være sættet af alle løsninger, hvor hver løsning i sættet er et sæt bestående af kursusnumre for alle valgte kurser for denne løsning.
Lad $ØK$ være sættet af kursusnumre for alle kurser der ønskes taget.
Lad $aøk(l, ØK)$ være en funktion, der giver antallet af opfyldte constraints fra $ØK$ i en løsning l.

$$\forall l \in L : aøk(l, ØK) \leq aøk(Iv, ØK)$$

Kurser der ikke må vælges på et bestemt semester, må ikke ligge på det angivne semester i studieplanen *(VK9)*:
Lad UK være sættet af alle par (k, s), hvor k er et kursusnummer for et kursus, der ikke må vælges, og s er det semester, det ikke må vælges på.
Lad $sem(sk)$ være en funktion, der giver det semester, som skemakode sk ligger på.

$$\forall (k, s) \in UK : (\forall i \in Iv : i = k \rightarrow sem(C_i) \neq s)$$

Kurser der ikke må vælges overhovedet, må ikke være en del af studieplanen *(VK9)*:
Lad UK være sættet af kursusnumre for alle kurser der ikke må vælges.

$$\forall i \in Iv : i \notin UK$$

Pointsummen af kurserne inden for et givent fagområde skal opfylde det givne krav *(VK10)*:
Lad F være sættet af alle par (f, p), hvor f er et fagområde og p er antallet af point, der mindst skal bruges.
Lad $f(i)$ være en funktion, der giver sættet af de fagområder som i tilhører.

$$\forall (f, p) \in F : \left(\forall i \in Iv \wedge f \in f(i) : \sum E(i) \geq p\right)$$

3.3.2 Constraints for en gyldig studieplan

Alle obligatoriske kurser for en given studieretning *skal* vælges *(S4)*:
Lad $ok(sr)$ være en funktion, der giver sættet af kursusnumre for alle obligatoriske kurser for studieretningen sr.
Lad asr være den aktuelle studieretning.

$$\forall i \in ok(asr) : i \in (Iv \cup TB)$$

3.3.2.1 Bachelor

En bachelor-studieplan skal bestå af kurser svarende til 180 ECTS-point *(S1)*:

$$\sum_{i \in Iv} E(i) = 180$$

En studieplan for en bachelor kan maksimalt bestå af 12 semestre *(S2)*:

$$n \leq 10$$

En bachelor-studieplan må ikke indeholde phd-kurser *(S6)*:
Lad PHD være sættet af kursusnumre for alle phd-kurser.

$$\forall i \in PHD : i \notin Iv$$

En bachelor-studieplan skal indeholde kurser svarende til minimum 45 ECTS-point inden for Naturvidenskabelige grundfag for en given studieretning *(S7)*:
Lad $ng(sr)$ være en funktion, der giver sættet af kursusnumre for kurser på studieretningen sr med kursustypen *Naturvidenskabeligt grundfag*.
Lad asr være den aktuelle studieretning.

$$\sum_{i \in (Iv \cap ng(asr))} E(i) \geq 45$$

En bachelor-studieplan skal indeholde kurser svarende til minimum 45 ECTS-point inden for Teknologiske linjefag for en given studieretning

***(S7)*:**
Lad $tl(sr)$ være en funktion, der giver sættet af kursusnumre for kurser på studieretningen sr med kursustypen *Teknologisk linjefag*.
Lad asr være den aktuelle studieretning.

$$\sum_{i \in (Iv \cap tl(asr))} E(i) \geq 45$$

En bachelor-studieplan skal indeholde kurser svarende til minimum 45 ECTS-point inden for Projekter og almene fag for en given studieretning *(S7)*:
Lad $pa(sr)$ være en funktion, der giver sættet af kursusnumre for kurser på studieretningen sr med kursustypen P*projekter og almene fag*.
Lad asr være den aktuelle studieretning.

$$\sum_{i \in (Iv \cap pa(asr))} E(i) \geq 45$$

Bachelor-projektet må ikke vælges før kurser svarende til minimum 90 ECTS-point er bestået *(S13)*:
Lad bac være kursusnummeret for bachelor-projektet.
Lad $ss(s)$ være en funktion, der giver den sidste skemakode i det forrige semester i forhold til s.

$$\sum_{i \in \{x \mid C_x \in [1;\, ss(C_{bac})]\}} E(i) \geq 90$$

3.3.2.2 Kandidat

En kandidat-studieplan skal bestå af kurser svarende til 120 ECTS-point *(S1)*:

$$\sum_{i \in Iv} E(i) = 120$$

En studieplan for en kandidat kan maksimalt bestå af 8 semestre *(S2)*:

$$n \leq 6$$

Der må ikke vælges grundlæggende kurser for mere end 10 ECTS-point *(S5)*:
Lad $gk(i)$ være en funktion, der giver en sandhedsværdi svarende til om i er et grundlæggende kursus.

$$\sum_{i \in \{x \mid x \in Iv,\, gk(x) = true\}} E(i) \leq 10$$

En kandidat-studieplan skal indeholde kurser inden for Generelle Retningskompetencer for en given studieretning svarende til det studieretningsspecifikke minimum af ECTS-point *(S8)*:
Lad $gr(sr)$ være en funktion, der giver sættet af kursusnumre for kurser på studieretningen sr med kursustypen *Generelle Retningskompetencer*.
Lad $p(t, sr)$ være en funktion, der giver det minimale antal ECTS-point for kursustypen t på studieretningen sr.
Lad asr være den aktuelle studieretning.

$$\sum_{i \in (Iv \cap gr(asr))} E(i) \geq p(asr)$$

En kandidat-studieplan skal indeholde kurser inden for Teknologisk Specialisering for en given studieretning svarende til det studieretningsspecifikke minimum af ECTS-point *(S8)*:
Lad $ts(sr)$ være en funktion, der giver sættet af kursusnumre for kurser på studieretningen sr med kursustypen *Teknologisk Specialisering*.
Lad $p(t, sr)$ være en funktion, der giver det minimale antal ECTS-point for kursustypen t på studieretningen sr.
Lad asr være den aktuelle studieretning.

$$\sum_{i \in (Iv \cap ts(asr))} E(i) \geq p(asr)$$

Kandidat-specialet må ikke vælges før kurser svarende til minimum 55 ECTS-point er bestået *(S13)*:
Lad kan være kursusnummeret for kandidat-specialet.
Lad $ss(s)$ være en funktion, der giver den sidste skemakode i det forrige semester i forhold til s.

$$\sum_{i \in \{x \mid C_x \in [1;\, ss(C_{kan})]\}} E(i) \geq 55$$

3.3.3 Constraints for en partiel studieplan

Det skal være muligt at udvide alle partielle studieplaner til gyldige studieplaner *(S14)*:
Lad L være sættet af alle løsninger, hvor hver løsning i sættet er et sæt bestående af kursusnumre for alle valgte kurser for denne løsning.
Lad $g(l, sr)$ være en funktion, der giver en sandhedsværdi svarende til om løsningen l er en gyldig studieplan for studieretningen sr.
Lad asr være den aktuelle studieretning.

$$\forall l \in L : (\exists I' \subseteq I : g(l \cup I', asr))$$

3.4 Omskrivning til lineær heltals aritmetik

Vi vil vise hvordan man kan omskrive den constraint, der sikrer at en gyldig bachelor uddannelse giver 180 ECTS point, til et udtryk i lineær heltals aritmetik. Med brug af teori om tabeller og mængdelære ser vores constraint således ud:

$$\sum_{i \in Iv} E(i) = 180$$

Her gør vi bl.a. brug af definitionen Iv, der dækker over det sæt af kurser der er valgt i studieplanen. Hvis man skal omskrive dette det til et udtryk uden brug af mængdelære, kan det gøres ved at udtrykke præcis hvad der skal gælde for hver eneste mulige kombination af valgte kurser i én stor konjunktion. Da antallet af kombinationer stiger eksponentielt med antallet af kurser, vil dette blive en uoverskuelig constraint at skulle udtrykke i lineær heltals aritmetik. For at demonstrere dette, vil vi vise hvordan constrainten ville se ud for et meget simpelt problem, hvor der kun er 3 mulige kurser at vælge.

Eksempel
Lad os antage at der kun findes følgende 3 kurser, med det angivne antal ECTS point som en konstant:
Kursus 1: Point1
Kursus 2: Point2
Kursus 3: Point3
Hvis kursusnumrene er henholdsvis 1, 2 og 3 vil det give følgende 3 variabler i vores system:

$$C_1, C_2, C_3$$

I en løsning hvor C_1 og C_3 er valgt vil det betyde at følgende gælder om variablernes værdi:

$$C_1 > 0, C_2 = 0, C_3 > 0$$

Når vi så her gerne vil udtrykke at denne løsning skal give 180 ECTS point, kan vi skrive det som følgende udtryk:

$$(C_1 > 0 \wedge C_2 = 0 \wedge C_3 > 0) \rightarrow (Point1 + Point3 = 180)$$

Her har vi udtrykt vores constraint for én mulig løsning. Den samlede constraint vil så blive samtlige mulige løsninger skrevet sammen i én stor konjunktion:

$$\begin{aligned}
&(C_1 = 0 \wedge C_2 = 0 \wedge C_3 = 0) \rightarrow false \\
\wedge&(C_1 > 0 \wedge C_2 = 0 \wedge C_3 = 0) \rightarrow (Point1 = 180) \\
\wedge&(C_1 = 0 \wedge C_2 > 0 \wedge C_3 = 0) \rightarrow (Point2 = 180) \\
\wedge&(C_1 = 0 \wedge C_2 = 0 \wedge C_3 > 0) \rightarrow (Point3 = 180) \\
\wedge&(C_1 > 0 \wedge C_2 > 0 \wedge C_3 = 0) \rightarrow (Point1 + Point2 = 180) \\
\wedge&(C_1 > 0 \wedge C_2 = 0 \wedge C_3 > 0) \rightarrow (Point1 + Point3 = 180) \\
\wedge&(C_1 = 0 \wedge C_2 > 0 \wedge C_3 > 0) \rightarrow (Point2 + Point3 = 180) \\
\wedge&(C_1 > 0 \wedge C_2 > 0 \wedge C_3 > 0) \rightarrow (Point1 + Point2 + Point3 = 180)
\end{aligned}$$

Ud fra dette eksempel ses det tydeligt hvor lange vores constraints ville blive, hvis de var udtrykt direkte i lineær heltals aritmetik uden brug af yderligere teorier. I og med alle vores constraints kan omskrives til denne form, vil det ikke blive et problem for vores program at evaluere dem.

3.5 Alternative SMT-reduktioner

Vi kunne også have valgt lave vores SMT-reduktion på andre måder.
Man kunne lade hver variabel repræsentere et semester, således at variablens værdi svarer til hvilke kurser der er taget på det givne semester. Dette er dog ikke en særlig elegant løsning, da antallet af de kombinationer kurserne kan udgøre er eksponentielt større end antallet af udbudte kurser.
En anden mulighed er at opdele hvert semester i alle de forskellige skemaplaceringer der er, og lade hver af disse skemaplaceringer være en variabel, vis værdi svarer til hvilket kursus der er taget på den givne skemaplacering. Dermed vil antallet af mulige værdier svare direkte til antallet af udbudte kurser. Et eksempel på en variabel med denne løsning kunne være:

$$V_{1,10} = \text{Matematik 1}$$

Det ville så betyde at kurset Matematik 1 er taget på første semester og ligger

på skemaplaceringen E3B. Dette kan dog simplificeres yderligere ved at lade variablens værdi være et heltal der svarer til et kursusnummer. Dermed vil eksemplet i stedet se således ud:

$$V_{1,10} = 01005$$

Hvis man også her lader værdien 0 svare til at intet kursus er valgt på det tidspunkt, vil man kunne repræsentere en partiel studieplan for ét semester som følgende række variable:

$$V_{1,1} = 0,\ V_{1,2} = 01005,\ V_{1,3} = 02021,\ \ldots,\ V_{1,n} = 0$$

Dette er en også god repræsentation af en studieplan. Vi valgte dog at bruge vores repræsentation frem for denne, da vores er tilsvarende simpel, og samtidig gør det nemmere at formulere bestemte constraints. Eksempelvis er det ikke nødvendigt at lave en constraint der udtrykker at et kursus ikke må vælges flere gange, da det allerede er en egenskab i repræsentationen.

Det er i denne sammenhæng også værd at overveje hvilken repræsentation der vil køre hurtigst når den er implementeret i Z3. Det er her afgørende både hvor mange variable der findes i systemet, og hvilken type de har. Z3 finder generelt hurtigere løsninger for boolean værdier end for tal. En mulig reduktion kunne derfor være at lade alle kurser være repræsenteret som et bitmap i systemet, hvor værdien i bitmappet svarer til om kurset er taget eller ej. Her vil man dog få det problem, at man ikke kan se ud fra et kursus' boolean værdi, hvilket semester kurset er taget på. Man kan dog løse dette problem, ved f.eks. at oprette flere værdier for hvert kursus, svarende til de forskellige semestre de kan ligge på. Denne repræsentation vil sandsynligvis få systemet til at finde en løsning hurtigere, da den benytter boolean værdier. Det er dog uvist om det vil give en mærkbar forskel, da man samtidigt også bliver nødt til oprette flere variable for hvert kursus. Vi valgte denne løsning fra, da den ikke virkede så ligetil at implementere som den vi valgte.

Kapitel 4

Design

Vi vil i dette kapitel beskrive hvordan vi har brugt vores constraints fra forrige kapitel til at implementere et system, der kan generere studieplaner. Vi vil herunder analysere hvilke problemer vil opstå ved implementeringen af vores constraints, og beskrive hvordan vi har løst disse problemer. Vi vil dernæst beskrive hvordan vi har modelleret de nødvendige informationer om kurser og studieretninger i F#, og hvilke væsentlige funktioner vores repræsentation gør brug af. Vi vil også beskrive hvordan vores system får adgang til informationer om kurser, og hvilke problemer vi har haft med designet af dette. Til sidst vil vi beskrive hvordan brugeren kan anvende programmet, og hvilke valg vi har truffet omkring brugerens input.

4.1 Constraint generation

Man kan reducere problemet med at generere en studieplan til at tildele en række variabler nogle værdier, der opfylder de constraints, som vi lavede i forrige afsnit. Disse constraints skal sørge for, at kravene til en studieplan og til programmet er overholdt, men er dog i sig selv ikke nok til at udforme en korrekt studieplan. Derfor har vi også lavet følgende constraints:

Valgte kurser skal have den korrekte skemakode:
Lad $sk1(s)$ være en funktion, der giver skemaplaceringen svarende til skemakoden s. Lad $sk2(i)$ være en funktion, der giver skemaplaceringen for kurset med kursusnummer i.

$$\forall i \in Iv : sk1(C_i) = sk2(i)$$

Nogle skemakoder udelukker hinanden:
Lad $u(s)$ være en funktion, der giver sættet af alle skemakoder som skemakode s udelukker.

$$\forall i \in Iv : (\forall j \in (Iv \setminus \{i\}) : C_i \notin u(C_j))$$

Hvis man tager udgangspunkt i vores constraints som de er formuleret i forrige kapitel, vil der kunne opstå en række problemer. Eksempelvis kan kurser der har flere forskellige skemaplaceringer være vanskelige at repræsentere med et heltal. Desuden kan man risikere, at proceduren til at finde de løsninger, der både har det maksimale antal ønskede frie skemaplaceringer og det maksimale antal ønskede kurser valgt, vil være meget langsom. Dette skyldes at man vil blive nødt til at se på alle mulige kombinationer mellem de to krav. Vi vil i de følgende afsnit redegøre for vores løsninger på disse problemer.

4.1.1 Kurser med flere skemaplaceringer

Det er muligt for at kursus at have mere end én skemaplacering. Eksempelvis har kurset Fysik 1 både E1B og F1B. Dette giver et problem for vores valg af variable i systemet, da hvert kursus kun bliver tildelt én heltalsværdi, svarende til den skemaplacering den ligger på i et vist semester. Dette kunne i teorien løses ved at lade hver mulig kombination af skemaplaceringer svare til en heltalsværdi. Dette er dog yderst upraktisk, da der ikke er noget klart maksimum antal af skemaplaceringer et kursus kan have, og dermed findes et uendeligt antal kombinationer. Selv hvis vi kendte det maksimale antal skemaplaceringer, ville der være et kæmpe antal kombinationer, i og med der findes 32 forskellige skemaplaceringer. For et maksimum på 4 skemaplaceringer ville det give 2^{32}, eller 4.294.967.296 forskellige kombinationer.
Den løsning vi har valgt til problemet, er oprette flere variable i systemet til de kurser der har flere skemaplaceringer. Dermed kan hver skemaplacering svare til én heltalsværdi per semester, hvilket gør mange constraints betydeligt nemmere at implementere i programmet. Denne løsning giver dog samtidig behov for et par nye constraints, der skal sikre at systemet behandler disse ekstra variable hensigtsmæssigt. Lad os bruge kurset Fysik 1 som eksempel:
I stedet for at have én variabel C_1 til at repræsentere dette kursus med både E1B og F1B som skemaplacering, har vi nu i stedet C_1 til at repræsentere E1B

og C_2 til F1B. Da det ikke er muligt at vælge kurset på kun den ene af de to placeringer, er det nødvendigt at lave følgende constraint:

$$(C_1 > 0) \leftrightarrow (C_2 > 0)$$

Dermed sikrer vi at kurset kun kan blive valgt, hvis det er taget på begge placeringer. Dette er dog ikke nok, da vi samtidig bliver nød til at sikre at C_2, der repræsenterer F1B, bliver taget på semesteret umiddelbart efter C_1. Hvis vi antager at heltalsværdien for F1B er 17 større end E1B, vil dette kunne udtrykkes med følgende constraint:

$$C_2 = C_1 + 17$$

Da værdien 0 betyder at kurset ikke er valgt, skal dette dog kun gælde hvis C_1 og C_2 er større end 0. Da den forrige constraint sikrer at de altid er det samtidigt, kan vi nøjes med at skrive $(C_1 > 0) \rightarrow (C_2 = C_1 + 17)$.

Dette gælder dog kun for kurser der ligger på flere skemaplaceringer samtidigt. Det er også muligt for et kursus at kunne vælges på én blandt flere kombinationer af skemaplaceringer. Eksempelvis har kurset Introduktion til statistik placeringen E3A *eller* F4A. Dette håndterer vi ligeledes ved at dele kurset op i to variable. Her skal de samme constraints dog ikke gælde, da kurset naturligvis ikke kan tages på begge placeringer. Dette kunne man sikre med følgende constraint:

$$\neg((C_1 > 0) \wedge (C_2 > 0))$$

Vi har dog i stedet valgt at implementere dette ved hjælp af vores eksisterende constraint for kurser med pointspærring. Det vil sige at vi i dette eksempel ville tilføje det delkursus som C_1 repræsenterer som værende pointspærrende for C_2, og omvendt. Dermed sikrer vores constraint for pointspærring at begge delkurser ikke begge kan blive valgt i én løsning.

4.1.2 Softconstraints

Det er muligt for brugeren at angive hvilke kurser der ønskes i studieplanen, og hvilke skemaplaceringer der ønskes fri. Vi betegner disse krav *softconstraints*, da det ikke er nødvendigt at opfylde dem for at generere en studieplan. Vi har dog valgt at vores system i stedet skal opfylde så mange softconstraints som muligt.
Afhængigt af hvor mange softconstraints der bliver angivet i systemet, kan det tage lang tid at finde den bedste løsning. Hvis antallet af angivne softconstraints er n, er der 2^n mulige kombinationer af softconstraints der skal prøves. For at reducere dette antal, har vi valgt at prioritere ønskede kurser højere end

skemaplaceringer. Det vil sige at vi først finder den største mængde af kurser vi kan opfylde, hvorefter kurserne i denne mængde bliver betragtet som hard-constraints. Først bagefter finder vi hvor mange skemaplaceringsønsker vi samtidig kan opfylde. Dette betyder at vi ikke nødvendigvis finder den optimale løsning, da det f.eks. er muligt at den optimale løsning opfylder ti soft-constraints, og vores metode f.eks. kun finder løsningen med syv ønskede kurser og to skemaplaceringer. Til gengæld har vi reduceret antallet af løsningskombinationer der skal prøves til $2^n + 2^m$ i stedet for 2^{n+m}, hvor n er antallet af kursusønsker og m er antallet af skemaplaceringsønsker. Worst-case køretiden er dog det samme i de tilfælde hvor der kun bliver angivet én type ønsker.
Denne prioritering betyder at vores constraint omkring ønskede skemaplaceringer, som blev formuleret i forrige kapitel, i stedet bør se sådan ud:

Det maksimalt mulige antal uønskede skemakoder skal være fri *(VK7)*:
Lad L være sættet af alle løsninger, hvor hver løsning er et sæt bestående af kursusnumre for alle valgte kurser for denne løsning, *og hvor L indeholder det maksimale antal ønskede kurser.*
Lad US være sættet af skemakoder der ønskes fri.
Lad $aus(l, US)$ være en funktion, der giver antallet af opfyldte constraints fra US i en løsning l.

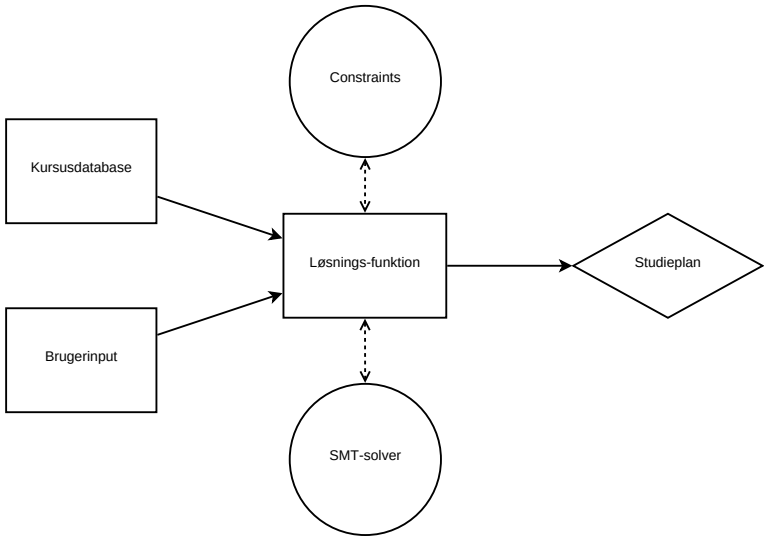
$$\forall l \in L : aus(l, US) \leq aus(Iv, US)$$

4.2 Model

For at kunne implementere vores system, har vi først taget stilling til hvordan vores system skal repræsenteres i F#.
Overordnet set har vi modelleret vores system, således at det består af fire forskellige dele:

- En række funktioner der svarer til alle vores constraints
- En database der indeholder den nødvendige information om alle kurser
- En funktion der indlæser og tolker brugerens input
- En løsningsfunktion der tager en given kontekst, og genererer en tilsvarende studieplan

Vi kan vise sammenhængen mellem disse dele med følgende tegning:



Figur 4.1

Vi vil i dette afsnit begrunde og beskrive hvordan vi har designet hver af disse dele i systemet.

4.2.1 Valg af typer i F#

Der er mange forskellige informationer vi skal kunne repræsentere i vores system, som f.eks. et kursus eller en skemaplacering. Vi har oprettet en lang række typer i F# til dette formål, og vi vil her beskrive og begrunde de vigtigste af typerne. Mange af de valgte typer dog er oplagte, som f.eks. at ECTS point har typen `float`, og vil derfor ikke blive begrundet.

Kursusnumre

```
type Coursenumber = string
```

Alle kursusnumre på DTU's kurser er tal på formen "xxxxx", og det vil derfor som udgangspunkt være naturligt at repræsentere dem som heltal i vores kode. I kursusbasen ligger der dog ikke kun DTU's kurser, men også kurser fra Københavns Universitet (KU). Disse kurser har kursusnumre på formen "KUxxx". Da vi ikke ville frasortere tilgængelige kurser fra kursusbasen, besluttede vi at repræsentere alle kursusnumre som tekststrenge.

Skemaplaceringer

```
type ScheduleCode =  |E1 |E2 ... |January |June |E |F |Unknown
```

Vi har valgt at typen `ScheduleCode` skal bestå af alle de primære og sammensatte skemaplaceringer nævnt i introduktionen samt `January` og `June` for 3-ugers perioden i henholdsvis januar og juni. Der findes også kurser, der ikke kan vælges på nogle af de traditionelle skemaplaceringer på et semester, men bare er defineret som liggende i enten efteråret og/eller foråret. Derfor har vi valgt også at tilføje `E` og `F`, der dækker over henholdsvis hele efterårs-semesteret og hele forår-semesteret. Til sidst har vi valgt også at tilføje `Unknown`, som alle kurser, der ikke kan vælges på nogle af de andre skemaplaceringer, får tildelt. Denne svarer derfor til den tomme skemaplacering.

Kursusforudsætninger

```
type CoursePrerequisites = Set<Set<CourseNumber>>
```

For at afbilde kursusbasens måde at skrive kursusforudsætninger på bedst muligt, har vi valgt at repræsentere disse som en konjunktion af disjunktioner. Det vil sige at et kursus med forudsætningerne (*A eller B*) *og* (*C eller D*) vil have værdien $\{\{A, B\}, \{C, D\}\}$.

Kurser

```
type Course = CourseNumber * CourseName * ECTS * CourseLevel *
ScheduleCode * PointBlock * MandatoryCoursePrerequisites * Fields
```

Som beskrevet i introduktionen skal vi udover kursusnumre, skemaplaceringer og kursusforudsætninger også bruge kursusnavn, antal ECTS point, kursusniveau, pointspærrede kurser og fagområder til at repræsentere et kursus. Vi skal desuden også bruge kursustype, men disse har vi i stedet valgt at repræsentere ved at hjælp af et sæt bestående af alle kursusnumre tilhørende kurserne med denne kursustype. Resten har følgende typer:

```
type CourseName = string
type ECTS = float
type CourseLevel = int
type PointBlock = Set<CourseNumber>
type Fields = Set<string>
```

Vi har valgt at repræsentere kursusniveauet som et heltal, da dette er lettere at bruge sammen med Z3.

Kursusdatabasen

```
type CourseDatabase = Course array
```

Den mest simple måde at repræsentere kursusdatabasen på, så den er let at slå op i, er ved en tabel (array).

Kurser med flere skemaplaceringer

```
type InputCourse = CourseNumber * CourseName * ECTS * CourseLevel *
Set<ScheduleCode> * PointBlock * MandatoryCoursePrerequisites * Fields
```

Da nogle kurser ligger på mere end én skemaplacering, indlæser vi først disse som typen `InputCourse`, der i stedet for én skemaplacering har et sæt af skemaplac-

eringer. De bliver senere delt op i flere kurser med én skemaplacering hver.

Studieretninger

```
type BachelorStudy = StudyCourses * MandatoryStudyCourses *
BasicScienceCourses * TechnologicalStudyCourses * ProjectsAndCommonCourses
type MasterStudy = StudyCourses * GeneralCompetenceCourses *
TechnologicalSpecializationCourses
```

Alle disse kursustyper er repræsenteret som sæt af kursusnumre. Vi har valgt at undlade kursustypen `MandatoryStudyCourses` fra kandidat-studieretningerne, og i stedet manuelt indtaste de obligatoriske kurser til hver retning. Dette har vi gjort da det i flere tilfælde viste sig at man ud fra en række kurser, selv kunne vælge hvilke kurser, der skulle være obligatoriske.

Studieplan

Vi har først og fremmest valgt at hvert semester skal repræsenteres således:

```
type Semester = Set<Course * int * ScheduleCode>
```

Den midterste type repræsenterer hvilket semester, kurset er valgt på.
Dernæst er en studieplan repræsenteret som en tabel af semestre således:

```
type StudyPlan = Semester array
```

På denne måde kan vi sætte alle semestre i rækkefølge og nemt vise hele studieplanen fra start til slut.

Map fra kursusnummer til indeks

```
type CourseToIndex = Map<string, int>
```

Dette map bruger vi til at koble vores kursusdatabase sammen med kursusvariablerne i systemet. Ved at slå op i dette map kan man ud fra et kursusnummer både finde det relevante kursus i kursusdatabasen men også den tilhørende kursusvariabel.

Opdelte kurser

```
type PartialCoursePairs = Set<int * ScheduleCode>
```

Vi har oprettet denne type, således at vi efter at have opdelt et kursus med flere skemaplaceringer, kan gemme de opdelte kursers indeks og skemaplacering i en værdi af denne type. Dette er nyttigt, da vi i en constraints skal sikre at ingen delkurser i et sådan sæt er taget uden resten.

4.2.2 Design af vigtige funktioner

Constraint-funktioner

Alle vores constraint-funktioner tager først og fremmest kursusvariablerne som argumenter. For at Z3 kan håndtere vores constraints, giver disse funktioner en `BoolExpr` tilbage, hvilket er en boolean-værdi i Z3, der svarer til om det vi un-

dersøger for er sandt. Det vil sige, at alle constraint-funktionerne har følgende type:

```
val FUNCTION_NAME : courseVariables:IntExpr [] -> ... -> BoolExpr
```

Et eksempel kunne være *bachelorPointConstraint*-funktionen, der undersøger om der er taget præcist 180 ECTS point på en bacheloruddannelse. Udover kursusvariablerne har den en række andre argumenter, som den bruger til udregningen. Den har derfor typen:

```
val bachelorPointConstraint :
      courseVariables:IntExpr [] -> ectsPoints:IntNum [] ->
      passedCourses:PassedCourses -> courseDatabase:CourseDatabase ->
      BoolExpr
```

Kurser med flere skemaplaceringer

Alle de kurser der skal tages på flere forskellige skemaplaceringer, har vi valgt at opdele i flere kurser, således at hvert af de opdelte kurser har én skemakode hver. Til hver gruppe af opdelte kurser har vi valgt at lave et sæt af typen `PartialCoursePairs`, der består af indeksene og skemakoderne for disse kurser. Derefter udvider vi vores kursusdatabase med alle de opdelte kurser, samt laver et stort sæt, der består af alle `PartialCoursePairs`. Alt dette gør vi i funktionen *extendDB*.

Som en af vores constraints har vi valgt at lave funktionen *partialCourseConstraint*, der undersøger om de opdelte kurser er taget korrekt. Det gør den ved at undersøge alle `PartialCoursePairs` i det store sæt. Hvis mindst ét af indeksene i et `PartialCoursePairs`-sæt tilhører et valgt kursus, skal alle de andre indeks også tilhøre valgte kurser. Da et kursus godt kan dække over flere semestre og perioder, skal funktionen også sørge for at de opdelte kurser er taget i den rigtige rækkefølge, samt inden for et naturligt interval. For eksempel skal der ikke være flere semestre imellem kurserne. Dette gøres blandt andet ved at udregne forskellen på deres skemakoder.

Løsnings-funktioner

Vi har valgt at dele vores system op i 3 forskellige studieplans typer. En partiel studieplan, en bachelor studieplan og en kandidat studieplan. Man kunne også have valgt at have en 4. type, der kunne svare til at planlægge en studieplan for hele civilingeniør uddannelsen. Vi valgte dog denne type fra, da det ikke afspejler den måde studerende normalt planlægger studiet, da de to uddannelser har forskellige studieretninger, og det ikke er muligt at blive optaget på kandidaten uden at have gennemført bacheloren.

Vi har valgt at lave en stor løsnings-funktion til hver studieplans-type. Hver af disse funktioner skal først om fremmest udvide kursusdatabasen, således at kurser med flere skemaplaceringer bliver delt op. Dernæst skal de oprette variable i systemet, svarende til kurserne i den udvidede database. Til sidst skal funktionen kalde alle de relevante constraint-funktioner, der er nødvendige for at generere en studieplan. Løsnings-funktionen for en bachelor-studieplan bruger

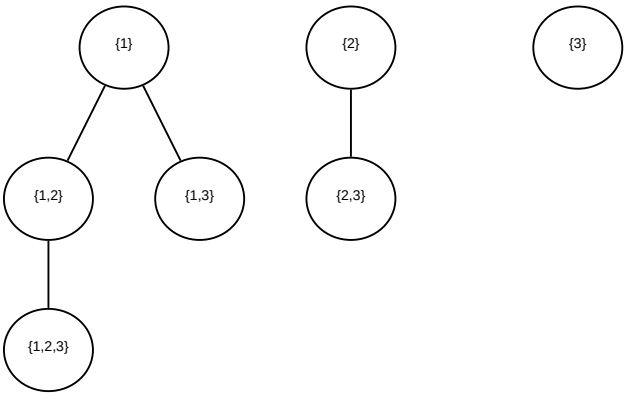
alle de generelle constraint-funktioner, samt dem der er unikke for bachelor-studieretningerne, men bruger ikke dem der er unikke for kandidat-studieretningerne. I løsningsfunktionen til en partiel studieplan bruger vi en constraint-funktion kaldet *expandableToValid*, der undersøger om en partiel studieplan kan udvides til enten en bachelor- eller en kandidat-studieplan ved at forsøge at løse en af disse.

Softconstraints
Vi har lavet en funktion til hver type softconstraint, der givet brugerens input, finder den største mængde af softconstraints der kan blive opfyldt.
Softconstraints er både ønskede kurser, og skemaplaceringer der ønskes fri. Vi ønsker at opfylde så mange af disse krav som muligt. Én måde at opnå dette på, er ved at udregne potensmængden af alle softconstraints, dvs. mængden af alle delmængder, således at man har samtlige kombinationer af softconstraints. Dernæst kan man tilføje én af disse delmængder til mængden af kurser der *skal* vælges, og forsøge at finde en løsning. Dette kan man så gøre for alle delmængder, indtil man har fundet den største delmængde der gav en løsning. Man kunne her med fordel sortere alle delmængderne efter størrelse, og starte enten fra toppen eller bunden. Hvis man starter med den største delmængde, ved man at den første løsning man finder er den bedste. Starter man med den mindste, er den bedste løsning den der ligger umiddelbart før den delmængde der ikke er en løsning for. Denne løsningsstrategi er dog ineffektiv på trods af sådanne optimeringer, da det kræver at man først finder potensmængden, og derefter potentielt søger efter en løsning én gang per delmængde.
I stedet for at finde den bedste løsning for en mængde af softconstraints ved at udregne potensmængden, har vi valgt at bruge algoritmen beskrevet i følgende pseudokode, hvor S er mængden af softconstraints, og L er en mængde af softconstraints der findes en løsning for:

$f(S, L) =$
  if $S = \{\}$
  then L
  else $L' =$
    if $L \cup \{e\}$ har en løsning
    then $f(S \setminus \{e\}, L \cup \{e\})$
    else L
  if $S \setminus L' = \{\}$
  then $max(L, L')$
  else $max(f(S \setminus \{e\}, L), L')$

Denne algoritme fungerer ved at tage hvert element e i en mængde af softconstraints S samt en eksisterende løsning L, og tjekker hvorvidt det er muligt at finde en løsning ved at tilføje e til L. Hvis dette er muligt, kalder vi funktionen rekursivt med $S \setminus \{e\}$ og $L \cup \{e\}$, og gemmer den bedste løsning fra

dette kald i *L'*. Vi undersøger hvorvidt der findes elementer i *S*, der ikke er en del af løsningen *L'*. Hvis dette ikke er tilfældet, er der ingen grund til at søge efter bedre løsninger for *S*. Derved forsøger vi ikke at finde løsningen {2,3} hvis løsningen {1,2,3} allerede er fundet. Vi kan vise hvordan algoritmen fungerer ud fra følgende tegning:



Figur 4.2

Hver knude i træet kan svare til én kombination af de tre softconstraints 1, 2 og 3. Algoritmen fungerer så ved at lave en dybde først søgning i træet. Hver gang det ikke er muligt at finde en løsning for en knude, vil algoritmen ignorere alle børn til knuden. Det vil sige at hvis {1,2} ikke er en løsning, vil knuden {1,2,3} blive sprunget over, og algoritmen i stedet kigge på {1,3}. Algoritmen vil også ignorere alle knuder i træet, hvor man ved at de ikke kan give en bedre løsning end den hidtil bedste. Det vil sige at hvis knuden {1,2,3} er en løsning, vil algoritmen stoppe så snart denne er fundet.

4.2.3 Studieretninger

Det er nødvendigt i vores system, at vi har en række informationer om studieretninger. Vi skal både vide hvilke kurser der hører til studieretningen, hvilke kurser der er obligatoriske, og hvilke kurser der har en given kursus type inden for studieretningen. Denne information ligger fri tilgængeligt i studiehåndbogen på tekstform, men at lave en funktion der kan indlæse dette, ville være i hel projekt i sig selv. Vi har derfor valgt at manuelt indtaste disse informationer i systemet. Da dette ville tage lang tid at gøre, har vi nøjes med at gøre det for bachelor-retningen Softwareteknologi og kandidat-retningen Informationsteknologi, da det ville være et kedeligt stykke arbejde at skulle manuelt indtaste

hundredvis af kurser for alle studieretningerne.
Der er også varierende krav der gælder for kandidat-retningerne. Eksempelvis kræver de fleste studieretninger at man skal tage for 30 ECTS point blandt kurserne med typen Generelle Retningskompetencer. Der er dog eksempler på studieretninger, hvor kravet i stedet er 35 point. Mange kandidat-studieretninger har også valgfrihed blandt de obligatoriske kurser - dvs. man eksempelvis **skal** tage enten kursus *A* eller kursus *B*. På grund af disse særtilfælde, har vi valgt at lave en constraint der gælder specifikt til hver kandidat-studieretning, og som sørger for at disse krav bliver opfyldt.

4.2.4 Specielle kurser

Det har været nødvendigt at indtaste et par kurser manuelt i systemet. For eksempel skal kurset Matematik 1 tages på forskellige skemaplaceringer, alt efter hvilken studieretning man er på. For at kunne tage højde for dette i systemet, har vi manuelt lavet 3 forskellige versioner af Matematik 1, svarende til de 3 forskellige placeringer kurset kan have. Bachelorprojektet og kandidatspecialet findes heller ikke som kurser i kursusbasen, så for at disse kan blive valgt som en del af studieplanen, har vi selv måtte oprette dem i systemet.

4.2.5 Udeladte funktionaliteter

Som det fremgår af vores kravspecifikation, havde vi oprindeligt tænkt, at hvert kursus skulle have tilknyttet en række fagområder, således brugeren af programmet kunne kræve at studieplanen skulle indeholde et antal ECTS point inden for bestemte fagområder. Kurserne i kursusbasen har dog desværre ingen fagområder tilknyttet, så vores system kan dermed ikke genkende hvilke kurser der tilhører hvilke fagområder. Vi har dog alligevel implementeret fagområder som en del af vores system, således at denne funktionalitet kan benyttes, hvis kursusbasen engang kom til at indeholde den nødvendige information.
Vi havde også tænkt at brugeren skulle have mulighed for at lade programmet tolke de såkaldte faglige forudsætninger som obligatoriske forudsætninger. Derved ville de valgte kurser i studieplanen have en bedre sammenhæng med hinanden. Det kunne dog ikke lade sig gøre, da de faglige forudsætninger i kursusbasen ikke bliver angivet som kursusnumre, men som beskrivende sætninger, hvor navne og kursusnumre kan indgå. Denne form gør det ikke muligt at tage højde for de faglige forudsætninger i programmet og vi må derfor undlade en sådanne funktion.

4.3 Database for kursusinformation

4.3.1 DTUs kursusbase

Det er nødvendigt at vores system har kendskab til samtlige udbudte kurser på DTU. Vi har derfor oprettet en database af typen *courseDatabase = course* `[]`, som vi kan slå op i for at finde den ønskede information om et kursus.
Det er muligt i DTUs kursusbase at hente hele kursuskataloget som en XML-fil. Vi har valgt at bruge denne XML-fil i programmet, til at oprette vores egen database for alle kurser. Her ses et lille udsnit af XML-filen for et kursus:

```
...
<Course CourseCode="01005" StartDate="2012-08-01Z"
 EndDate="2013-07-31Z" University="DTU" Version="11"
 LastUpdated="2012-04-26Z" CourseType="DTU_REGULAR">
        <Description_Language Lang="dk"/>
        <Description_Language Lang="en-gb"/>
        <Teaching_Language LangCode="dk" AlsoTaughtInEnglish="false">
          <Description KeyValue="dk" Lang="dk" Txt="Dansk"/>
          <Description KeyValue="dk" Lang="en-gb" Txt="Danish"/>
        </Teaching_Language>
        <Title Lang="en-gb"
         Title="Advanced Engineering Mathematics 1 "/>
        <Title Lang="dk" Title="Matematik 1 "/>
        ...
```

Som det kan ses her, indeholder XML-filen en lang række informationer som vi ikke er interesseret i, som f.eks. start- og slutdato. I dette udsnit er de eneste informationer vi skal bruge kursets nummer og navn. Disse informationer optræder her under attributterne *CourseCode* og *Title*. Ud fra dette udsnit ses det altså, at kursets nummer er *01005* og at navnet på dansk er *Matematik 1*.

4.3.2 Parser-funktionen

Vi har lavet en funktion *readXmlFile*, der tager en XML-fil som argument, og giver en kursusdatabase tilbage i form af et InputCourse-array. Da en XML-fil i F# er det samme som en string, vil funktionen have typen:
`val readXmlFile : string -> InputCourse []`
Denne funktion læser igennem hele XML-filen, og finder alle de informationer der

skal bruges i typen InputCourse for samtlige kurser. For kurser vis skemaplacering der indeholder *eller*, ønsker vi at dele disse op, således at et kursus med kursusnummeret *xxxx* og med skemaplaceringen "E1A eller F1A", bliver til kurserne *xxxxA* med placeringen "E1A" og *xxxxB* med placeringen "F1A". Vi tilføjer her et ekstra bogstav på kursusnummeret, således vi kan kende forskel på kurserne, og dermed sørge for *xxxxA* og *xxxxB* pointspærrer for hinanden. Dette introducerer dog et nyt problem, da kurset alle andre steder i systemet stadig har nummeret *xxxx*, f.eks. i andre kursers forudsætninger.
Dette problem har vi løst ved at oprette et map af typen
`CourseNumberMap = Map<CourseNumber, Set<CourseNumber>>`, som mapper alle kursusnumre til den mængde af nye kursusnumre vi har opdelt dem i. Dermed kan vi slå *xxxx* op i mappet, og få mængden {*xxxxA*,*xxxxB*}. Da vores funktion også skal returnere dette map, betyder det at funktionen i stedet har typen:

```
val readXmlFile : string -> InputCourse [] * CourseNumberMap
```

4.3.3 Antallet af kurser i kursusdatabasen

Vores oprindelige tanke var at oprette en variabel i systemet for hvert eneste kursus i kursusbasen. Vi har dog måtte indse efter implementeringen, at så stort et antal variable får programmets køretid til at blive alt for stor. Vi har derfor valgt kun at oprette variable for de kurser der faktisk har mulighed for at blive valgt i den aktuelle kørsel. Det vil sige alle de kurser der hører til den aktuelle studieretning, samt de kurser brugeren angiver i inputtet. Denne mængde af kurser giver vi med i vores parser-funktion som argument, således vi kan sortere alle andre kurser i kursusbasen fra. I praksis vil det sige at vi har færre end hundrede kursusvariable i systemet, hvor vi før havde over tusind. Typen for vores endelige version af parser-funktionen efter denne ændring er dermed:

```
val readXmlFile : string -> Set<CourseNumber> ->
                  InputCourse [] * CourseNumberMap
```

4.3.4 Uoverensstemmelser og generelle problemer

Da alle kurser i DTUs kursusbase bliver oprettet manuelt af forskellige kursusansvarlige, er den fyldt med fejl og uoverensstemmelser. Udenfor vores system er det ikke et problem, da de studerende har let ved at gennemskue hvad meningen er. Et eksempel på dette er kurset Indledende programmering med Matlab, hvor skemaplaceringen står som E1B **og** F2B, hvor det burde være **eller**. Da kurset kun er 5 ECTS point kan alle studerende regne ud, at det ikke kører på begge semestre. Denne fejl vil dog gøre at vores system bruger kurset forkert,

således at en studieplan genereret af vores system risikerer ikke at være gyldig. Der er også en generel uoverensstemmelse i kursusbasen omkring brugen af "/" og "." ved pointspærring og forudsætninger. De to tegn bruges i stedet for *og* og *eller*, men bliver brugt forskelligt fra kursus til kursus. Dette kan særligt ses ved pointspærring, hvor de kurser der bliver opremset altid skal tolkes med et *og* imellem, men hvor både "/" og "." bliver brugt i forskellige kurser. Vi har derfor besluttet at vores parser-funktion altid tolker "/" som *eller*, og "." som *og* for kursusforudsætninger, og tolker både "/" og "." som *og* for pointspærring. På grund af denne uoverensstemmelse i kursusbasen, tolker vores system altså halvdelen af alle kursusforudsætninger forkert.

4.4 Bruger-interaktion

Vi vil her beskrive hvilke valg vi har truffet omkring brugerens input til systemet, samt beskrive hvordan vi har designet brugerens anvendelse af programmet.

4.4.1 Standardværdier for brugerinput

Vi har givet brugeren en lang række muligheder for at stille krav til programmet. Det er selvfølgelig ikke nødvendigt at benytte alle disse muligheder for at planlægge et studie, og vi har derfor været nød til at tage stilling til, hvordan vi tolker de krav-muligheder, som brugeren har undladt at benytte. Et eksempel på et sådant undladt krav, kan ses under ønskede kurser i input-eksemplet fra forrige afsnit, hvor ingen kurser er angivet.
For de fleste undladte krav giver det sig selv hvordan systemet skal tolke dette. Hvis en bruger eksempelvis ikke benytter kravet om kurser der skal vælges, tolker systemet naturligvis dette som om ingen krav om valgte kurser skal stilles til studieplanen. Ligeledes, hvis en bruger angiver et ønsket kursus, men ikke angiver på hvilket semester det ønskes, tolker vi dette som om kurset må vælges på et vilkårligt semester. Denne tolkning er helt naturlig, men ikke alle krav-muligheder er så ligetil.
Hvis en bruger eksempelvis ikke angiver et minimalt og maksimalt antal ECTS point per semester, giver det ikke mening at tillade systemet at vælge kurser for vilkårligt mange point. Vi har derfor valgt at dette bliver tolket som 30 point, hvilket er det normerede antal for et semester. Ligeledes med kravet om 3-ugers perioder, har vi valgt at der skal vælges et kursus i hver 3-ugers periode, hvis brugeren ikke stiller krav til dette, da 13-ugers perioden ellers ville blive overnormeret.
Samtlige standardværdier kan ses i kravspecifikationen for hver inputtype.

4.4.2 Tidligere beståede kurser

Én af de ting en bruger kan angive i programmet, er tidligere beståede kurser, i tilfælde af at brugeren allerede er i gang med uddannelsen. Dermed kan systemet regne ud hvor mange ECTS point der allerede er optjent, hvilke kursusforudsætninger der allerede er opfyldt, og hvilke kurser der er pointspærret for. Det er dog ikke alle kurser der skal give både ECTS point, tælle som kursusforudsætninger og spærre for andre kurser. Hvis man eksempelvis ønsker at planlægge et kandidat studie, og allerede har gennemført en bachelor, skal alle tidligere beståede kurser gælde som kursusforudsætninger og lave pointspærring, men uden at give point. Hvis man derimod er i gang med kandidatuddannelsen og ønsker at planlægge resten, skal alle kurserne tælle som kursusforudsætninger og lave pointspærring, men kun kurserne fra kandidaten skal give point.
Vi har derfor valgt at brugeren skal angive de tidligere beståede kurser i to forskellige kategorier, bachelor- og kandidatkurser. Dermed kan vi i systemet tage højde for præcis hvordan kurserne skal håndteres, alt efter hvilken situation brugeren er i. Alternativt kunne man lade brugeren angive hvor mange ECTS point der allerede er optjent, og lade alle tidligere beståede kurser blot tælle som kursusforudsætninger og pointspærringer.

4.4.3 Anvendelse af programmet

Det optimale for brugeren ville være, hvis alle krav til systemet kunne angives i en grafisk brugerflade. Dermed kunne man lave, så brugeren vælger den ønskede studieretning eller antallet af semestre, ud fra en liste af gyldige valgmuligheder. Vi har dog valgt en grafisk brugerflade fra, da dette vil tage lang tid at lave, og vi i stedet har valgt at fokusere vores arbejde på det bagvedliggende system.
Vi har derfor valgt at brugerens krav bliver angivet i en kommasepareret tekstfil, som vores system kan indlæse og oversætte til de passende typer i systemet. Et eksempel på en sådan fil kunne være følgende med forklaring ud fra hvert input:

`Gyldig`	Studieplanstype (Gyldig/Partiel)
`Softwareteknologi`	Studieretning
`6`	Studieplanslængde
`25-30,30,30-35,30,30,25`	Pointfordeling på semestre
`Efterår`	Årstid for studiestart (Efterår/Forår)
`ja,nej,nej,ja,ja,nej`	Fordeling af 3-ugers kursus på semestre
`02101,01005`	Kurser der skal vælges
	Kurser der ønskes valgt
`01007`	Kurser der ikke må vælges
`E1-1,F3-3,E1B-2`	Skemaplaceringer der skal være fri på semesteret
`E2A-1`	Skemaplaceringer der ønskes fri på semesteret
`30-4`	Udlandsophold med angivet point på semesteret
	Tidligere bestående kurser på bacheloren
	Tidligere bestående kurser på kandidaten

Figur 4.3

Kapitel 5

Z3

Z3 er en SMT-solver der bliver udviklet af Microsoft Research. Vi har valgt at bruge Z3 i vores opgave fremfor andre SMT-solvers, da vi på forhånd havde kendskab til Z3 fra et tidligere kursus.

5.1 SMT-solving

En SMT-solver fungerer ved at kombinere en almindelig SAT-solver med forskellige Theory-solvers. Lad os antage at vi har følgende problem i lineær heltals aritmetik:

$$a > 0 \wedge b > 0 \wedge (c = a + b \vee c = 0)$$

En SMT-solver vil forsøge at løse dette problem ved først at omskrive hvert led i problemet til en atomisk proposition. I dette eksempel vil der blive oprettet disse fire propositioner:

$$p_1 \equiv (a > 0), p_2 \equiv (b > 0), p_3 \equiv (c = a + b), p_4 \equiv (c = 0)$$

Med disse omskrivninger kan problemet nu udtrykkes som følgende SAT-problem:

$$p_1 \wedge p_2 \wedge (p_3 \vee p_4)$$

Dette problem bliver så løst i SAT-solver delen, som forsøger at finde en model til dette problem. Hvis det ikke kan lade sig gøre, kan man allerede her konkludere at det oprindelige problem ikke kan løses. I dette eksempel, kunne SAT-solveren f.eks. finde denne model:

$$p_1, p_2, p_3, p_4$$

Disse propositioner vil så blive oversat igen, således at vi har disse fire aritmetiske udtryk:

$$a > 0, b > 0, c = a + b, c = 0$$

Dette problem vil blive løst i en Theory-solver, der i dette tilfælde er en solver der bruger teori om lineær heltals aritmetik. Da der ikke findes en model for dette problem, vil Theory-solveren sige at $a > 0, b > 0, c = a + b, c = 0$ er unsatisfiable. Dette giver anledning til følgende lemma, der udtrykker at alle fire propositioner ikke kan være sande på samme tid:

$$\neg p_1 \vee \neg p_2 \vee \neg p_3 \vee \neg p_4$$

Dette lemma vil så blive tilføjet til det oprindelige SAT-problem, således at det bliver udvidet til følgende:

$$p_1 \wedge p_2 \wedge (p_3 \vee p_4) \wedge (\neg p_1 \vee \neg p_2 \vee \neg p_3 \vee \neg p_4)$$

Dette problem vil så pånyt blive løst i SAT-solveren, og løsningen vil igen blive oversat til aritmetik og løst i Theory-solveren. Sådan vil SMT-solveren blive ved indtil der enten er fundet en løsning, eller det ikke længere er muligt at finde en model i SAT-solveren.
I dette konkrete eksempel, vil SAT-solveren f.eks. kunne finde modellen $p_1, p_2, \neg p_3, p_4$, hvilket bliver oversat til $a > 0, b > 0, \neg(c = a + b), c = 0$. Dette vil Theory-solveren kunne finde en model til, eksempelvis $a = 1, b = 2, c = 0$.

5.2 Brug af Z3 i F#

Vi vil her vise hvordan man kan bruge Z3 sammen med F# til at løse et konkret problem.
Lad os tage udgangspunkt i problemet fra forrige afsnit. Det første vi skal gøre for at løse dette problem, er at oprette en context for de tre variable a, b og c:

```
let ctx = new Context()
```

Dernæst skal de tre variable oprettes som heltalsvariable i denne context. Dette gøres med funktionen `MkIntConst`:

```
let a = ctx.MkIntConst("a")
let b = ctx.MkIntConst("b")
let c = ctx.MkIntConst("c")
```

Dernæst opretter vi de tre constraints $a > 0$, $b > 0$ og $c = a + b \vee c = 0$ i contexten. Til dette bruger vi funktionerne `MkEg`, `MkGt`, `MkAdd` og `MkOr`, til at repræsentere symbolerne $=$, $>$, $+$ og $\vee$ henholdsvis. Tallet 0 skriver vi ved hjælp af funktionen `MkInt`:

```
let constraint1 = ctx.MkGt(a,ctx.MkInt 0)
let constraint2 = ctx.MkGt(b,ctx.MkInt 0)
let constraint3 = ctx.Or(ctx.MkEq(c,ctx.MkAdd(a,b)))
```

Det sidste vi nu mangler er, at tjekke hvorvidt der findes en fortolkning af disse variable, der opfylder de tre constraints. For at gøre dette opretter vi en solver i contexten:

```
let solver = ctx.MkSolver()
```

For at sikre at de tre constraints bliver opfyldt i fortolkningen, bruger vi funktionen `Assert` på de tre constraints:

```
solver.Assert(constraint1)
solver.Assert(constraint2)
solver.Assert(constraint3)
```

Dernæst beder vi programmet om at finde en løsning med funktionen `Check()`:

```
let solution = solver.Check()
```

Til sidst tjekker vi at løsningen er satisfiable, og skriver løsningen ud med `solver.Model` hvis det er tilfældet:

```
match solution with
| Status.SATISFIABLE -> printfn "Resultat: %A" solver.Model
| _ -> printfn "Ingen løsning"
```

Kapitel 6

Implementering

I dette kapitel vil vi beskrive hvordan vi har implementeret vores system. Vi vil starte med at give et overblik over strukturen af implementeringen, samt beskrive hvordan de forskellige dele af koden hænger sammen med vores design. Derefter vil vi give en mere detaljeret beskrivelse af de forskellige dele af koden, med udgangspunkt i de mere komplicerede funktioner. Til sidst vil vi analysere på køretiden af vores system, og undersøge hvilke funktioner der er flaskehals for systemet.

6.1 Kodestruktur

Vi har i implementeringen valgt, for overskuelighedens skyld, at opdele vores program i flere filer. Herunder vil vi give et overblik over de forskellige filer, samt en kort beskrivelse af deres indehold:

Constraints
Se koden i Appendiks A.
Det er her alle vores constraint-funktioner er definerede. Man kan desuden opdele denne gruppe i fire undergrupper: Constraint-funktioner for generelle studieplaner, constraint-funktioner for bachelor-studieplaner, constraint-funktioner

for kandidat-studieplaner og øvrige constraint-funktioner.
Eftersom man på flere kandidatretninger selv kan vælge, hvilke kurser der skal være obligatoriske ud fra en række af kurser, opretter vi også specifikke constraints for obligatoriske kurser til hver kandidatretning. Der er også forskellige krav til mængden af point, man skal have inden for de andre kursustyper, så derfor opretter vi også specifikke constraints til dem.

Model
Se koden i Appendiks B.
Her har vi alle konstruktions-funktionerne, bl.a. funktionerne der opretter kursusvariablerne og udvider kursusdatabasen med de opdelte kurser, som beskrevet tidligere, samt de tre løsnings-funktioner.
Vi definerer desuden alle de typer, som programmet skal bruge, f.eks. typen *Naturvidenskabelige grundfag* (*BasicScienceCourses*) er et sæt af kursusnumre (*Set<CourseNumber>*). Derudover definerer vi også tre omregningsfunktioner: *codeToInt* der konverterer en skemaplacering til en skemakode, *intToCode* der konverterer en skemakode til en skemaplacering og *stringToCode* der konverterer en skemaplacering på tekstform til en skemaplacering.

Kursusdatabasen
Se koden i Appendiks C.1.
Implementeringen af kursusdatabasen består af en stor funktion, der, ud fra en xml-fil med kursusbasen, finder alle relevante kurser og opretter vores kursusdatabase. Det er også i denne funktion, at kurser med valgfrie skemaplaceringer bliver opdelt.
Vi opretter desuden manuelt nye kurser for bl.a. bachelorprojekterne og et nyt kursus for hver gruppe af specifikke skemaplaceringer.

Studieretningsdatabase
Se koden i Appendiks C.2.
Her opretter vi manuelt de forskellige kursustyper til hver studieretning, f.eks. opretter vi kursustypen *Projekter og almene fag* for studieretningen *Softwareteknologi* således:

```
softwareTechnologyProjectsAndCommonCourses =
   set ["02101";"02121";"02122";"42610"]
```

Det er også her vi laver databaserne over bachelor- og kandidat-studieretningerne samt et map, der kan fortælle om en studieretning er en bachelor-studieretning eller en kandidat-studieretning ud fra navnet på studieretningen.

Bruger-interaktion
Se koden i Appendiks D.
Vi har her lavet en funktion, der tager brugerens input fra en tekstfil og laver

det om til de typer, som den relevante løsnings-funktion skal bruge.

6.2 Constraint generering

Vores constraints er implementeret som en række funktioner, der alle tager et array af Z3 variable som argument, og returnerer en boolean-værdi af typen `BoolExpr`. Afhængigt at hvilket krav vi undersøger i en constraint, kan funktionen også tage flere argumenter, svarende til de informationer der er nødvendige at kende. Grunden til at funktionerne alle returnerer en `BoolExpr` er, at *Assert()*-funktionen i Z3, som vi bruger til at overholde vores constraints, kun tager argumenter af denne type.
Vi vil her beskrive de constraint-funktioner der var mest vanskelige at implementere: Funktionen der sikrer at opdelte kurser ikke tages uafhængigt af hinanden, funktionerne der finder den største mængde af soft constraints vi kan opfylde, og funktionen der sikrer at en partiel studieplan kan udvides til en gyldig.

partielCourseConstraint
Se koden i Appendiks A.4.
I denne funktion skal vi sørge for at de opdelte kurser bliver valgt i den rigtige rækkefølge og med den korrekte afstand mellem deres skemakoder. Til dette bruger vi et stort sæt, hvor hvert element i dette sæt er et sæt bestående af indekset og skemakoden for alle de opdelte kurser, der hører sammen. For hvert sæt af indeks og skemakoder bliver alle sådanne par sammenlignet med alle andre par i sættet ved hjælp af en funktion.
Denne funktion får to argumenter, der hvert består af et indeks og en skemakode. Begge skemakoder bliver så konverteret til det tilsvarende heltal ved hjælp af vores *codeToInt*-funktion, hvorefter forskellen på dem udregnes. Herefter gælder det, at hvis et af kurserne er valgt, så skal de begge ligge på skemakoder, hvor forskellen svarer til den tidligere udregnede forskel.

Soft constraints
Der er to typer soft constraints, som vi skal behandle, soft constraints for ønskede kurser og soft constraints for uønskede skemaplaceringer. Vi har lavet to næsten identiske funktioner, der hver især finder den største mængde af begge typer soft constraints. Begge funktioner er lavet på baggrund af algoritmen, som vi formulerede i design kapitlet. Vi vil her kigge på implementeringen af funktionen *courseSoftConstraint*:

```
val courseSoftConstraint : CourseWantedSoftConstraints ->
CourseWantedHardConstraints -> CourseWantedHardConstraints
```

Funktionen tager et mængde af ønskede kurser *cwscs*, og en mængde af ønskede

kurser *solutionSet*, der tolkes som værende hard constraints. Ud fra disse mængder, returneres den største mængde af ønskede kurser der kan opfyldes.
Funktionen fungerer ved at vælge et element *e* fra mængden *cwscs*, og hvorefter den undersøger hvorvidt der findes en løsning, hvis *e* tilføjes til *solutionSet*. Hvis der findes en løsning, kalder vi funktionen rekursivt, hvor *e* er fjernet fra *cwscs*, og tilføjet til *solutionSet*. Denne løsning kalder vi *solutionSet'*. Hvis ikke, undersøger vi om der findes elementer i *cwscs*, som ikke er en del af den bedste løsning vi fandt i det rekursive kald. Hvis sådanne elementer findes, kalder vi igen funktionen rekursivt, hvor *e* er fjernet *cwscs*. Hvis ikke, returnerer vi den bedste løsning blandt *solutionSet* og *solutionSet'*.
Vi kalder funktionen med mængden af samtlige ønskede kurser, og den tomme mængde af kurser der har en løsning, hvorefter de rekursive kald sørger for at prøve med alle de nødvendige kombinationer af soft constraints.

expandableToValid
Se koden i Appendiks A.4.
Denne funktion bruges til at undersøge om en partiel studieplan kan udvides til en gyldig bachelor- eller kandidat-studieplan. Dette gør den ved, ud fra inputtet fra den partielle løsnings-funktion, at kalde den af de gyldige løsnings-funktioner, der er relevant for den givne studieretning. Her ændrer vi dog på forskellige input, således at de passer med en gyldig løsning. Eksempelvis ændrer vi på antallet af semestre i studieplanen, således at bachelor- og kandidat-løsningen bliver fundet med hhv. 12 og 8 semestre. Hvis en sådan løsning findes, undersøges der om alle valgte kurser fra den partielle løsning også er valgte i den gyldige løsning. Hvis dette ikke er tilfældet er det ikke en udvidelse af den partielle løsning.

6.3 Model

Ud over alle vores constraints, består vores implementering af en løsnings-funktion for hver type studieplan, samt en række konstruktions-funktioner som disse løsnings-funktioner benytter. Løsnings-funktionerne fungerer alle ved at tage brugerens input og en kursusdatabase som argumenter, og genererer en studieplan på baggrund af disse. Vi vil her forklare hvordan løsnings-funktionerne er implementeret, med udgangspunkt i funktionen *solveValidBachelor*, der genererer gyldige studieplaner for bacheloruddannelsen. Vi vil desuden forklare hvordan den vigtigste af konstruktions-funktionerne *extendDB* er implementeret.

solveValidBachelor
Se koden i Appendiks B.3.
Databasen som denne funktion får givet, har typen *InputCourse* `[]`. Det vil sige

at kurser i denne database hver har et sæt af skemaplaceringer. Det første funktionen gør er, at lave en ny database med typen `CourseDatabase = Course []`, hvor hvert kursus kun har én skemaplacering. Dette sker i funktionen *extendDB*, som vi blive forklaret senere i afsnittet. Dernæst blive der oprettet et array af heltals-variable i Z3 svarende hvert kursus i den nye database.
Det næste der sker i funktionen er, at vi finder det største mængde af ønskede kurser vi kan opfylde, ved hjælp af funktionen *courseSoftConstraints*. Denne mængde af kurser bliver så tilføjet til de kurser der **skal** vælges. Ligeledes finder vi den største mængde af uønskede skemaplaceringer vi kan opfylde. Her sørger vi for at bruge den nye mængde af kurser der **skal** vælges, således at skemaplaceringerne bliver prioriteret efter kurserne.
Efter at have fundet de soft constraints vi kan opfylde, bruger vi Z3-funktionen *Assert()* til at sikre at alle constraints der er relevante for en bacheloruddannelse bliver overholdt. Dernæst undersøger vi hvorvidt det er muligt at finde en løsning. Hvis ikke, returnerer funktionen `None`. Hvis der kunne findes en løsning, finder vi hvilket semester hver af de valgte kurser ligger på, således vi kan returnere en løsning af typen `StudyPlan = Semester []`. Desuden returnerer vi en model af vores løsning i Z3, da vi skal bruge denne løsning i funktionen *expandableToValid*.

extendDB

Se koden i Appendiks B.2.

```
val extendDB : InputCourseDatabase -> CourseDatabase * Set<PartialCoursePairs>
```

Denne funktion tager en database med typen `InputCourse []` som argument, og returnerer en udvidet database, hvor alle kurser med flere skemaplaceringer er blevet delt op. Desuden returnerer den en mængde af typen `PartialCoursePairs`, der indeholder alle de opdelte kurser, angivet med indeks og skemaplacering. Disse mængder af opdelte kurser skal vi bruge i constrainten *partialCourseConstraint*, hvor vi sørger for at alle delkurser for et kursus med flere skemaplaceringer bliver valgt samtidigt.
Funktionen gør brug af to hjælpefunktioner, *makeLists* og *splitCourse*. I funktionen *makeLists* løber vi rekursivt igennem den oprindelige database, og kalder *splitCourse* hvert kursus. I *splitCourse* løber vi rekursivt igennem alle kursets skemaplaceringer, opretter et delkursus for hver af dem, og tilføjer delkursernes indeks og skemaplacering til en mængde af typen `PartialCoursePairs`. Disse delkurser bliver så tilføjet forrest til en foreløbig liste af kurser i *makeLists*, og deres indeks og skemaplacering tilføjet til en foreløbig mængde af `PartialCoursePairs`. Hvis *splitCourse* bliver kaldt på et kursus med én skemaplacering, bliver kurset blot tilføjet til listen af kurser. Når *makeLists* har været igennem alle kurser, returneres den nye liste af kurser, samt mængden af `PartialCoursePairs`. Da alle kurser blev tilføjet forrest til listen af kurser, sørger vi for at vende listen om, således rækkefølgen passer med indekset i mængden af `PartialCoursePairs`.

6.4 Database for kursusinformation

Implementeringen af vores database for kursusinformation foregår udelukkende i funktionen *readXmlFile*.

readXmlFile
Se koden i Appendiks C.1.1.

```
val readXmlFile : string -> Set<CourseNumber> ->
(InputCourse [] * CourseNumberMap)
```

Funktionen tager to argumenter, en XML-fil i form af en string, og et sæt af kursusnumre vi ønsker at have i databasen. Først og fremmest vil vi læse igennem stregen. Til dette opretter vi en StringReader til at kunne læse igennem strengen gradvist, hvorefter vi bruger denne StringReader til at oprette en XmlReader. Dermed kan vi genkende XML-filens struktur mens vi læser igennem den, og bruge XmlReaderens indbyggede funktioner, som f.eks. *GetAttribute()*, til at finde de kursusinformationer vi ønsker.
Vi kalder den indlejrede funktion *read*, og bruger den til at læse igennem hele XML-filen. Funktionen *read* returnerer en liste af kurser, samt en liste af oprindelige kursusnumre med deres tilsvarende nye numre. Når vi har fået returneret disse, tilføjer vi de specielle kurser fra filen *specialCourses.fs* til begge lister, hvorefter vi returnerer kursuslisten som et array og kursusnummer-listen som et map.

read

```
val read : XmlReader -> List<InputCourse> ->
List<CourseNumber * Set<CourseNumber>> ->
(List<InputCourse> * List<CourseNumber * Set<CourseNumber>>)
```

Funktionen *read* tager tre argumenter, en XmlReader *reader*, en foreløbig liste af kurser i databasen *courseList*, og en foreløbig liste over hvilke forskellige numre kurserne har *courseNumberList*. Den sidstnævnte liste skal vi bruge, da vi giver kurser med valgfrie skemaplaceringer nye kursusnumre, og ønsker at lave et map fra det gamle kursusnummer til de nye. Funktionen fungerer ved at kalde sig selv rekursivt indtil den indbyggende XmlReader-funktion *Read()* ikke længere er sand. Hver gang vi støder på et element med navnet "Courses" i XML-filen, opretter vi en ny XmlReader med *ReadSubtree()*, og kalder funktionen *readCourse* med den nye XmlReader for at få kursets information. Når vi så kender kursets information, undersøger vi hvorvidt kurset har valgfrie skemaplaceringer. Hvis det er tilfældet, kalder vi funktionen *splitDisjointScheduleCodes*, der sørger for at dele kurset op i flere og tilføje hver til *courseList*, samt tilføje de nye kursusnumre til *courseNumberList*. Hvis kurset ikke har valgfrie skemaplaceringer, bliver det blot tilføjet til *courseList* og *courseNumberList*.
readCourse

`val readCourse : XmlReader -> InputCourse -> InputCourse` + Funktionen *readCourse* finder den nødvendige kursusinformation i XML-filen. Den tager to argumenter, en XmlReader *reader* og et foreløbigt kursus *inputCourse*. Den fungerer ved at bruge *reader* læse igennem alt den information XML-filen indeholder omkring et bestemt kursus. Hver gang *reader* støder på et element der indeholder en information vi skal bruge, f.eks. kursets navn, kaldes en relevant hjælpefunktion der kan finde informationen i elementet. Denne information tilføjes så til *inputCourse*, og *readCourse* bliver kaldt rekursivt med den opdaterede kursusinformation. Når *reader* kommer til slutningen af XML-filen for kurset, returneres *inputCourse*, der så vil indeholde alle de nødvendige informationer.

6.5 Bruger-interaktion

Efter at brugeren har angivet sine krav i en tekstfil, bruger vil funktionen *generateStudyPlan* til at generere den ønskede studieplan. Denne funktion gør desuden brug af funktionen *readInput*, der indlæser brugerens krav fra tekstfilen, og konverterer kravene til de aktuelle typer i vores system. Se koden i Appendiks D.

generateStudyPlan

```
val generateStudyPlan : string -> string -> StudyPlan
```

Denne funktion tager filnavnet på brugerens inputfil samt filnavnet på den ønskede output som argumenter, og returnerer en studieplan.
Funktionen fungerer ved at indlæse brugerens input ved hjælp af funktionen *readInput*. Dernæst finder funktionen mængden af relevante kurser for den givne studieretning, og opretter en kursusdatabase ved hjælp af funktionen *readXmlFile*. Til sidst kaldes den relevante løsnings-funktion for det givne input, med kursusdatabasen og alle brugerens krav som argumenter. Denne løsning gemmes i en tekstfil med det ønskede navn ved hjælp af en *StreamWriter*.

readInput

```
val readInput : string -> StudyType * StudyName * StudyLength
* ECTSDistribution * SemesterStartInSpring * ThreeWeekPeriodConstraints
* CourseConstraints * ScheduleCodeConstraints * StudyAbroadSemesters
* PassedCourses * PassedCourses
```

Denne funktion tager et filnavn som argument, og giver filens indhold tilbage som en række typer vi kan bruge i systemet.
Funktionen fungerer ved at først at læse hele filens indhold til en string. Da hvert krav i filen er opdelt med tegnet ";", bruger vi *Split*-funktionen til at lave et array, hvor hvert element i arrayet er en string der repræsenterer et krav. Hvert af disse elementer konverterer vi til den rette type, og returnerer til sidst

alle de konverterede krav.

6.6 Køretidsanalyse af vores constraint-funktioner

Køretiden af et satisfiability problem udtrykt i lineær heltals aritmetik vokser eksponentielt med problemets størrelse. Vores system er derfor meget følsom over for constraints der tager lang tid at verificere.
Langt de fleste constraint-funktioner har en worst case kompleksitet på $O(n)$ hvor n er antallet af heltals variabler, da vi ofte blot løber vores variabler igennem én gang, og laver forskellige tjek i konstant tid. Andre funktioner, som f.eks *pointBlock*, har worst case kompleksiteten $O(n * m)$, hvor m er det gennemsnitlige antal kurser der bliver pointspærret for. Worst case kan m være $n - 1$, hvilket ville give worst case kompleksiteten $O(n^2)$, men i praksis er m tæt på én for vores kurser, og den forventede kompleksitet er derfor $\Theta(n)$.
Funktionen *courseOverlapConstraint* har dog kompleksiteten n^2, da vi for hvert kursus undersøger at alle de andre kurser ikke overlapper med dennes skemaplacering. Denne constraint forøger dermed køretiden af vores system drastisk. Eksempelvis tog det 10 sekunder at generere en studieplan ud fra en bestemt kontekst, hvis denne constraint ikke blev brugt, og 4 minutter at generere en studieplan ud fra samme kontekst, hvor constrainten blev brugt. Dette hænger dog også sammen med, at denne constraint drastisk begrænser antallet af løsninger for et givent problem.
To andre constraints der har vist sig at forøge køretiden af vores system tilsvarende drastisk, er de constraints der sikrer at pointsummen for en gyldig bachelor- og kandidat-studieplan er overholdt. Hvorfor dette er tilfældet, er vi dog ikke sikre på, da kompleksiteten af disse constraint kun er $O(n)$. En mulig forklaring kunne være, at denne constraint også drastisk begrænser antallet af løsninger til problemet. Vi har dog forsøgt at teste dette ved at undlade denne constraint, og i stedet sikre den rette pointsum ved f.eks. at sige hvert semester i en studieplan med seks semestre, skal indeholde 30 point. Et sådant problem løser vores system markant hurtigere, på trods af at antallet af løsninger er det samme, og vi ved derfor ikke hvorfor disse sum-constraints påvirker den samlede køretid så markant.

Kapitel 7

Test

Vi vil i dette kapitel beskrive to forskellige typer test af vores system, eksterne og interne test. Først vil vi lave eksterne test, hvor vi undersøger hvorvidt vores system opfylder de programkrav *K1-4* som vi formulerede i kravspecifikationen. Dernæst vil vi lave interne test, hvor vi undersøger hvorvidt udvalgte funktioner i vores program fungerer korrekt. Vi har her valgt ikke at teste samtlige funktioner, og i stedet blot demonstrere fremgangsmåden ved hjælp af de udvalgte funktioner.

7.1 Eksterne test

Vi vil i dette afsnit teste hvorvidt vores system opfylder de fire programkrav (*K1-4*) som er formuleret i kravspecifikationen.

7.1.1 Programkrav K1

Vi vil her blot teste at vores system kan indlæse en brugers input fra en tekstfil. For at opfylde dette krav, har vi implementeret funktionen *readInput*. For at teste denne funktion har vi oprettet en tekstfil med en kontekst "K1Test.txt",

hvorefter vi kalder *readInput* med filnavnet "K1Test.txt" som argument, og undersøger hvorvidt de værdier funktionen returnerer, svarer til den angivne kontekst i filen. Tekstfilen og funktionens output kan ses i Appendiks E.1.1.
Ud fra dette output, kan vi se at funktionen har returneret den givne kontekst som en række værdier vi kan tolke i systemet. Eksempelvis er teksten "Gyldig" blevet konverteret til værdien *Valid* af typen `StudyType`, og teksten "ja,ja,ja,nej,ja,nej" er blevet konverteret til et array af typen `bool`.

7.1.2 Programkrav K2

Vi vil her teste at systemet kan generere en gyldig studieplan, på baggrund af en given kontekst. For at den genererede studieplan er gyldig, skal den opfylde kravene *S1-13* i definitionen af en gyldig studieplan. Desuden skal studieplanen tage højde for alle informationer og opfylde alle krav angivet i den givne kontekst, *OI1-2*, *VI1-2* og *VK1-11*. Vi vil teste dette ved at definere en kontekst i systemet, og undersøge hvorvidt en studieplan genereret på denne baggrund opfylder de nødvendige krav. Studieplanen og konteksten kan ses i Appendiks E.1.2.

S1: Dette krav er opfyldt, da der er valgt kurser for 90 ECTS point i studieplanen, og der er angivet et udlandsophold for yderligere 30 point i konteksten.

S2: Dette krav er opfyldt, da studieplanen strækker sig over 4 semestre.

S3: Dette krav vil altid være opfyldt, da der skal angives en studieretningen i konteksten for at en studieplan kan blive genereret.

S4: De obligatoriske kurser der skal være valgt for studieretningen Informationsteknologi, er enten "Virksomhedsstart" eller "Teknologi, økonomi, ledelse og organisation", samt kandidat speciale. Dette krav er opfyldt, da "Teknologi, økonomi, ledelse og organisation" og "Speciale" begge er valgt.

S5: Dette krav er opfyldt, da der ikke er valgt et eneste grundlæggende kursus.

S6-7: Disse krav gør sig kun gældende for bacheloruddannelser, og er derfor altid opfyldt for studieretningen Informationsteknologi.

S8: Dette krav er opfyldt, da der både er valgt generelle retningskompetencer og teknologiske specialiseringskurser for 30 ECTS point.

S9: Dette krav er i teorien opfyldt, da kurset "Praktisk digital konstruktion" som optræder flere gange i studieplanen, også ligger på flere skemaplaceringer i kursusbasen. I praksis er kravet dog ikke opfyldt, dette skyldes en fejl i DTUs kursusbase.

S10: Dette krav er opfyldt, da den eneste skemaplacering der optræder flere gange, nemlig F, kun ligger på én gang på 2. og 4. semester.

S11: Dette krav er opfyldt, da de eneste kurser denne studieplan pointspærrer for, er kurserne med nummerene *02137*, *02221* og *02222*, og ingen af disse optræder i studieplanen.

S12: Dette krav er opfyldt, da det eneste kursus med obligatoriske forudsætninger er Data-logik med forudsætningen "Logiske systemer og logik programmering". Denne forudsætning er opfyldt, da dette kursus er angivet i konteksten som et tidligere bestået kursus.

S13: Dette krav er opfyldt, da kandidat specialet er valgt på 4. semester, hvor der er taget kurser for 60 ECTS point samt et udlandsophold på 30 point inden.

OI1: Der bliver taget højde for denne information, da der i konteksten bliver krævet en studieplan på 4 semestre.

OI2: Der bliver taget højde for denne information, da studieplanen opfylder kravene *S1-13* for studieretningen Informationsteknologi.

VI1: Der bliver taget højde for denne information, da der ingen kurser er valgt på 3 semester, svarende til det angivne udlandshold i konteksten.

VI2: Der bliver taget højde for denne information, da kurset 02156 er angivet i konteksten, og dette er forudsætning for det valgte kursus "Data-logik".

VK1: Dette krav er opfyldt, da alle kravene for en gyldig studieplan *S1-13* er opfyldt.

VK2: Dette krav er opfyldt, da det i konteksten er angivet at 3-ugers perioden skal vælges på 1. og 4. semester, hvilket er opfyldt af "Praktisk digital konstruktion" og "Speciale".

VK3: Dette krav er opfyldt, da det er krævet i konteksten at alle semestre skal indeholde 30 ECTS point, og der er valgt kurser for 30 ECTS point på alle semestre, undtagen semesteret der er angivet som udlandsophold for 30 point.

VK4: Dette krav er opfyldt, da det er krævet at studieplanen skal starte om efteråret, og alle valgte kurser på 1. semester er taget om efteråret.

VK5-6: Dette krav er opfyldt, da det er krævet at skemaplaceringen E1A skal være fri på 1. semester og ønsket at F1A er fri på 2. semester, hvilket er opfyldt i studieplanen.

VK7-9: Dette krav er opfyldt, da det i konteksten bliver krævet at "Speciale" skal tages på 4. semester, "Data-logik" skal tages på et vilkårligt semester, "Teknologi, økonomi, ledelse og organisation" ønskes taget på et vilkårligt semester og "Beregningsmæssigt hårde problemer" må ikke tages. Alle disse krav er opfyldt i studieplanen.

VK10: Dette krav er opfyldt, da der her ikke er angivet nogle krav til fagområder. Systemet ville dog ikke være i stand til at opfylde kravet hvis det blev angivet, da der som beskrevet ikke er tilstrækkelig information i kursusdatabasen.

Vores system opfylder dermed programkravet *K2*.

7.1.3 Programkrav K3

Vi vil her teste at systemet kan generere en partiel studieplan, på baggrund af en given kontekst. Det eneste krav en studieplan skal opfylde for at være partiel, er at den skal kunne udvides til en gyldig studieplan. Studieplanen skal også overholde de krav der er angivet i den kontekst den bliver genereret på baggrund af, men da disse krav er implementeret ens for partielle og gyldige studieplaner, vil vi ikke teste disse igen. I stedet vil vi teste at den genererede studieplan overholder kravet *S14*. Vores test kan ses i Appendiks E.1.3.
Vi tester *S14* ved at først at generere en partiel studieplan på 2 semestre. Derefter forsøger vi at generere en gyldig studieplan på 4 semestre, hvor alle valgte kurser i den partielle studieplan er angivet som tidligere beståede kurser. Denne gyldige studieplan vil dermed fungere som en mulig udvidelse af den partielle studieplan. I og med systemet kan finde en sådan udvidelse, ved vi at de to semestre i den partielle studieplan kan udvides til en gyldig studieplan, hvis man tilføjer de kurser systemet fandt for mulige udvidelse.

7.1.4 Programkrav K4

Dette krav har vi allerede vist er opfyldt i afsnittet om *K2* og *K3*, da disse test netop tager udgangspunkt i en gemt studieplan af denne type. De to gemte studieplaner kan ses hhv. i Appendiks E.1.2 og Appendiks E.1.3.

7.2 Interne test

Vi vil i dette afsnit teste at vores funktioner er implementeret korrekt, og returnerer den forventede værdi. Da vi har så mange funktioner, vil vi dog kun tage udgangspunkt i et par udvalgte, og ud fra disse vise hvordan man generelt kan teste funktionerne.

7.2.1 Constraint-funktioner

Alle vores constraint-funktioner returnerer en værdi af typen `BoolExpr`, hvilket er en Z3 type. Vi er ikke i stand til at evaluere værdier af denne type, da de variabler som optræder i en værdi af typen `BoolExpr` ikke får tillagt en værdi før Z3 forsøger at søge efter en løsning. Det vil sige at et kald af en constraint-funktion eksempelvis vil returnere `(= C_1 C_2)` i stedet for at evaluere hvorvidt variablerne C_1 og C_2 er ens.
På grund af dette er vi nødt til at teste vores constraints ved at give funktionerne en række variable som vi på forhånd kender værdien på, hvorefter vi manuelt må evaluere den værdi som funktionen returnerer. Eksempelvis kan vi beslutte at variablerne C_1 og C_2 har værdierne `MkInt 1` og `MkInt 2`, hvilket betyder at vi kan evaluere `(= (C_1) (C_2))` til boolean værdien $1 = 2 \equiv \mathit{false}$. For at vi realistisk kan evaluere sådanne værdier manuelt, er vi nødt til at arbejde få variabler ad gangen, da constraint-funktionerne ellers vil returnere enorme udtryk.

Vi vil i vores test af constraint-funktioner bruge følgende kursusdatabase som input:

Kursusnummer	Kursusnavn	Point	Niveau	Placering	Pointspærring
"No1"	"Kursus1"	5.0	1	E1B	set["No2"]
"No2"	"Kursus2"	15.0	1	F3	set["No1"]
"No3"	"Kursus3"	7.5	2	E1	set[]
"No4"	"Kursus4"	5.0	1	E2B	set["No5"]
"No5"	"Kursus5"	7.5	2	F5A	set["No4"]

Figur 7.1

Vi vil desuden tage udgangspunkt i en studieplan på to semestre med *Kursus1* som et tidligere bestået kursus.

7.2.1.1 Pointspærring

Vi vil her teste vores constraint-funktion der sikrer at ingen valgte kurser pointspærrer for hinanden. Vi har lavet fire forskellige tabeller af heltalsværdier som vi tester funktionen *pointBlock* med. Resultatet af alle testkørsler kan ses i Appendiks E.2.2.1.

Test 1
Vi benytter følgende værdier for de fem kurser:

C_1	C_2	C_3	C_4	C_5
1	5	7	2	10

Omskrevet til et almindeligt logisk udtryk giver funktionen følgende:

$$(IF\ 1 > 0\ \ THEN\ 5 = 0 \wedge \neg false\ \ ELSE\ true) \wedge \ldots$$

Allerede efter først led kan vi se at udtrykket er falsk. Dette er også funktionens forventede output, da alle fem kurser er valgt, på trods af at både *Kursus1* og *Kursus2*, og *Kursus4* og *Kursus5* spærrer for hinanden.

Test 2
Vi benytter følgende værdier for de fem kurser:

C_1	C_2	C_3	C_4	C_5
0	0	4	0	0

Omskrevet til et almindeligt logisk udtryk giver funktionen følgende:

$$(IF\ 0 > 0\ \ THEN\ 0 = 0 \wedge \neg false\ \ ELSE\ true)$$

$$\wedge(\ IF\ 0 > 0\ \ THEN\ 0 = 0 \wedge \neg true\ \ \ ELSE\ \ true)$$

$$\wedge(\ IF\ 4 > 0\ \ THEN\ true\ \ ELSE\ true)$$

$$\wedge(\ IF\ 0 > 0\ \ THEN\ 0 = 0 \wedge \neg false\ \ ELSE\ true)$$

$$\wedge(\ IF\ 0 > 0\ \ THEN\ 0 = 0 \wedge \neg false\ \ ELSE\ true)$$

Dette udtryk er sandt, hvilket også er funktionens forventede output, da det eneste kursus der er valgt ikke pointspærrer for andre kurser.

Test 3
Vi benytter følgende værdier for de fem kurser:

C_1	C_2	C_3	C_4	C_5
0	0	0	0	0

Omskrevet til et almindeligt logisk udtryk giver funktionen følgende:

$$(IF\ 0 > 0\ \ THEN\ 0 = 0 \wedge \neg false\ \ ELSE\ true)$$

$$\wedge(\ IF\ 0 > 0\ \ THEN\ 0 = 0 \wedge \neg true\ \ ELSE\ true)$$
$$\wedge(\ IF\ 0 > 0\ \ THEN\ true\ \ ELSE\ true)$$
$$\wedge(\ IF\ 0 > 0\ \ THEN\ 0 = 0 \wedge \neg false\ \ ELSE\ true)$$
$$\wedge(\ IF\ 0 > 0\ \ THEN\ 0 = 0 \wedge \neg false\ \ ELSE\ true)$$

Dette udtryk er sandt, hvilket svarer til funktionens forventede output, da en studieplan uden valgte kurser altid vil opfylde vores krav om pointspærring.

Test 4 Vi benytter følgende værdier for de fem kurser:

C_1	C_2	C_3	C_4	C_5
0	3	0	9	0

Omskrevet til et almindeligt logisk udtryk giver funktionen følgende:

$$(IF\ 0 > 0\ \ THEN\ 0 = 0 \wedge \neg false\ \ ELSE\ true)$$
$$\wedge(\ IF\ 3 > 0\ \ THEN\ 0 = 0 \wedge \neg true\ \ ELSE\ true)$$
$$\wedge(\ IF\ 0 > 0\ \ THEN\ true\ \ ELSE\ true)$$
$$\wedge(\ IF\ 9 > 0\ \ THEN\ 0 = 0 \wedge \neg false\ \ ELSE\ true)$$
$$\wedge(\ IF\ 0 > 0\ \ THEN\ 9 = 0 \wedge \neg false\ \ ELSE\ true)$$

Dette udtryk er falsk, hvilket svarer til funktionens forventede output. Dette skyldes at *Kursus1* pointspærrer for *Kursus2* på trods af at *Kursus1* ikke er valgt, da *Kursus1* er angivet som et tidligere bestået kursus.

7.2.1.2 Antal semestre

Vi vil her teste vores constraint-funktion der sikrer at alle valgte kurser ligger inden for det angivne antal semestre. Vi har lavet fire forskellige tabeller af heltalsværdier som vi tester funktionen *rightAmountOfSemesters* med. Resultatet af alle testkørsler kan ses i Appendiks E.2.2.2.

Test 1
Vi benytter følgende værdier for de fem kurser:

C_1	C_2	C_3	C_4	C_5
0	5	0	18	35

Omskrevet til et almindeligt logisk udtryk giver funktionen følgende:

$$(0 \leq 34 \wedge 0 \geq 0) \wedge (5 \leq 34 \wedge 5 \geq 0) \wedge (0 \leq 34 \wedge 0 \geq 0)$$
$$\wedge(18 \leq 34 \wedge 18 \geq 0) \wedge (35 \leq 34 \wedge 35 \geq 0)$$

Dette udtryk er falsk, hvilket svarer til funktionens forventede output, da *Kursus5* er valgt med værdien 35, svarende til at kurset ligger på 3. semester.

Test 2
Vi benytter følgende værdier for de fem kurser:

C_1	C_2	C_3	C_4	C_5
55	78	35	42	65

Omskrevet til et almindeligt logisk udtryk giver funktionen følgende:

$$5 \leq 34 \wedge 55 \geq 0) \wedge (78 \leq 34 \wedge 78 \geq 0) \wedge (35 \leq 34 \wedge 35 \geq 0)$$

$$\wedge(42 \leq 34 \wedge 42 \geq 0) \wedge (65 \leq 34 \wedge 65 \geq 0)$$

Dette udtryk er falsk, hvilket svarer til funktionens forventede output, da samtlige kursers værdi ligger uden for intervallet for 2 semestre.

Test 3
Vi benytter følgende værdier for de fem kurser:

C_1	C_2	C_3	C_4	C_5
1	5	7	2	10

Omskrevet til et almindeligt logisk udtryk giver funktionen følgende:

$$(1 \leq 34 \wedge 1 \geq 0) \wedge (5 \leq 34 \wedge 5 \geq 0) \wedge (7 \leq 34 \wedge 7 \geq 0)$$

$$\wedge(2 \leq 34 \wedge 2 \geq 0) \wedge (10 \leq 34 \wedge 10 \geq 0)$$

Dette udtryk er sandt, hvilket svarer til funktionens forventede output, da alle valgte kurser har en værdi svarende til 1. semester. Vi er i denne constraint ligeglade med at 2. semester er tomt for kurser.

Test 4
Vi benytter følgende værdier for de fem kurser:

C_1	C_2	C_3	C_4	C_5
1	5	28	18	34

Omskrevet til et almindeligt logisk udtryk giver funktionen følgende:

$$(1 \leq 34 \wedge 1 \geq 0) \wedge (5 \leq 34 \wedge 5 \geq 0) \wedge (28 \leq 34 \wedge 28 \geq 0)$$

$$\wedge(18 \leq 34 \wedge 18 \geq 0) \wedge (34 \leq 34 \wedge 34 \geq 0)$$

Dette udtryk er sandt, hvilket svarer til funktionens forventede output, da dette svarer til en typisk studieplan, hvor der er valgt kurser på samtlige mulige semestre.

7.2.2 Opdeling af kurser med flere skemaplaceringer

Vi vil her teste at vores funktion *extendDB* kan tage en tabel af kurser med et vilkårligt antal skemaplaceringer, og returnere en tabel, hvor alle kurser kun har én skemaplacering og et tilsvarende antal ECTS point. Funktionen skal desuden returnere en mængde af typen `PartialCoursePairs`, svarende til de kurser der er blevet opdelt. For at kunne teste dette, har vi oprettet en input-kursusdatabase, som vi kalder funktionen med. Dette test database svarer til indholdet af følgende tabel:

Kursusnummer	Kursusnavn	Point	Niveau	Placering	Pointspærring
"No1"	"Kursus1"	5.0	1	set[E1B]	set["No2"]
"No2"	"Kursus2"	15.0	1	set[F3A;F4A;F3B]	set["No1"]
"No3"	"Kursus3"	7.5	2	set[E1]	set[]
"No4"	"Kursus4"	5.0	1	set[E2B]	set["No5"]
"No5"	"Kursus5"	7.5	2	set[F5A;F4B]	set["No4"]

Figur 7.2

Vi tester funktionen ved kalde den med denne database. Funktionens output kan ses i Appendiks E.2.3. Den kursusdatabase som funktionen returnerer kan vi så stille op på samme form som inputdatabasen:

Kursusnummer	Kursusnavn	Point	Niveau	Placering	Pointspærring
"No1"	"Kursus1"	5.0	1	E1B	set["No2"]
"No2"	"Kursus2"	5.0	1	F3A	set[]
"No2"	"Kursus2"	5.0	1	F4A	set[]
"No2"	"Kursus2"	5.0	1	F3B	set["No1"]
"No3"	"Kursus3"	7.5	2	E1	set[]
"No4"	"Kursus4"	5.0	1	E2B	set["No5"]
"No5"	"Kursus5"	7.5	2	F5A	set[]
"No5"	"Kursus5"	7.5	2	F4B	set["No4"]

Figur 7.3

Vi kan her se at de kurser som på forhånd kun havde én skemaplaceringer optræder uændret i tabellen. Derimod er *Kursus2* og *Kursus5* blevet delt op i flere delkurser, således *Kursus2* eksempelvis nu er delt op i tre delkurser, hver med én skemaplacering og en tredjedel af det samlede kursus' ECTS point. Man kan her også se at det kun er ét af de ter delkurser der har det oprindelige kursus' pointspærring. Dette betyder dog intet, da vi alligevel binder disse delkurser sammen, således at enten alle eller ingen af dem er valgt.

Funktionen returnerer desuden et sæt hvor hvert element er en række indeks og skemaplaceringer, svarende til de bindinger der skal laves mellem delkurser. I denne konkrete test kan dette output vises i følgende tabel:

PartialCoursePairs
set[(1, F3A); (2, F4A); (3, F3B)]
set[(6, F5A); (7, F4B)]

Figur 7.4

Vi kan her se at hver af de nye delkurser der skal laves bindinger i mellem, optræder i samme element i sættet. Funktionen har dermed korrekt opdelt kurser med flere skemaplaceringer, således at vi har en ny udvidet database, samt en række delkurser der hænger sammen.

Kapitel 8

Anvendelse

Vi vil i dette kapitel give et simpelt eksempel på hvordan en bruger kan anvende vores system, og på baggrund af dette give et eksempel på en genereret studieplan. Vi vil dernæst beskrive hvilke problemer der er med anvendelsen af vores system, særligt i forhold til køretiden.

8.1 Eksempel på anvendelse af systemet

Vi vil give et eksempel på hvordan vores systemet kan anvendes. Lad os antage følgende:
En bruger af systemet skal starte på studieretningen Informationsteknologi om efteråret, og ønsker at gennemføre uddannelsen på normeret tid. Brugeren ønsker desuden at fordele alle ECTS point ligeligt ud på de forskellige semestre, og vil derfor også have et kursus i alle 3-ugers perioder. Brugeren opretter så den tilsvarende kontekst i en tekstfil med navnet *input.txt*:

```
Gyldig
Informationsteknologi
4
30,30,30,30
Efterår
ja,ja,ja,ja
.
.
.
```

```
.
.
.
.
.
EOF
```

Derefter kalder brugeren funktionen *generateStudyPlan* med argumenterne "input" og "output": `generateStudyPlan "input" "output";;`

Dette genererer følgende tekstfil *output.txt*:

```
Semester 1:
02165 - Udvikling af softwareprodukter - January
02209 - Test af digitale systemer - E1B
02264 - Kravspecifikation - E3
42490 - Teknologi, økonomi, ledelse og organisation - E5
ECTS point: 30

Semester 2:
02220 - Distribuerede systemer - F1B
02233 - Netværkssikkerhed - F4A
02238 - Biometriske systemer - June
02244 - Sprog-baseret sikkerhed - F2A
02291 - System integration - F5A
ECTS point: 30

Semester 3:
Mas30 - Speciale - January
Mas30 - Speciale - E
ECTS point: 30

Semester 4:
01410 - Kryptologi 1 - F2A
02105 - Algoritmer og datastrukturer 1 - F2B
02282 - Algoritmer for store datamængder - F1A
02285 - Kunstig intelligens og multiagent-systemer - F4A
30510 - GPS, GIS og afsætning for civilingeniører - June
ECTS point: 30

-----------------------------------------------------
Denne studieplan er genereret ud fra følgende input:
Gyldig
Informationsteknologi
4
30,30,30,30
Efterår
ja,ja,ja,ja
.
.
.
.
.
.
.
.
EOF
```

På baggrund af denne studieplan kan brugeren så tage stilling til, om studieplanen er tilfredsstillende, eller om der skal foretages ændringer. Hvis brugeren

eksempelvis hellere vil skrive kandidatspeciale på 4. semester, skal der blot tilføjes "Speciale-4" i tekstfilen *input.txt*, hvorefter *generateStudyPlan* skal kaldes igen.

8.2 Praktiske problemer ved anvendelse

Det største problem vi har med anvendelsen af vores system, er den tid det tager at generere en studieplan. Eksempelvis tog det vores system over 50 minutter at generere studieplanen fra forrige afsnit, og det tager endnu længere tid at generere gyldige bachelor-studieplaner. Da der sandsynligvis ikke er nogen der i praksis vil vente så længe for at generere en studieplan, er dette et problem der skal løses hvis der nogensinde skulle videreudvikles på vores system.
En anden hindring for anvendelsen af systemet, er valget om at håndtere input og output som tekstfiler i stedet for at gøre dette igennem en grafisk brugergrænseflade. Dette er bevidst valg fra starten af projektet, for i stedet at fokusere på selve systemet, men er også med til at demotivere eventuelle brugere fra at benytte systemet.

Kapitel 9

Videreudvikling

Vi vil her beskrive hvordan man med videre arbejde kunne løse nogle af de problemer der er i vores system.

9.1 Forbedringer af systemets køretid

Som vores system er nu, er det ikke realistisk at bruge i praksis, grundet systemets køretid. Dette kan forbedres en smule ved at lave små optimeringer af vores constraint-funktioner, eksempelvis ved at sørge for at systemet ikke løber så mange gange igennem arrayet med kursusvariablerne. Dette vil dog sandsynligvis ikke gøre nogen mærkbar forskel.
For at få systemet til at køre radikalt hurtigere, er det nok nødvendigt at lave systemet på baggrund af en anden SMT-reduktion. Eksempelvis kunne man lave en reduktion hvor kurserne er repræsenteret ved hjælp af boolean værdier i stedet for heltal, som beskrevet i afsnit 3.5.

9.2 Forbedringer af kursusbasen

Et af problemerne i vores system er de mange fejl og uoverensstemmelser i kursusbasen. På grund af disse, kan man ikke være sikker på at den genererede studieplan opfylder de nødvendige krav i virkeligheden. En oplagt forbedring ville derfor være at få rettet disse fejl i kursusbasen.
Det ville også være en væsentlig forbedring af vores system, hvis man fagområder og faglige forudsætninger stod angivet i kursusbasen på en form vores system kunne bruge. Dermed ville vores allerede implementerede funktionalitet omkring krav til fagområder kunne bruges, og man ville nemt kunne implementere de faglige forudsætninger på samme måde som vi har implementeret de obligatoriske.
Det er dog kun DTUs administration der har mulighed for at lave sådanne ændringer i kursusbasen, så det er ikke en videreudvikling vi kan gøre noget ved.

9.3 Brugervenlighed af systemet

Ud over systemets køretid, er manglen på brugervenlighed en stor forhindring i at vores system kunne bruges i praksis. En oplagt løsning af dette kunne være, at integrere vores system med en grafisk brugergrænseflade. Dermed vil det være muligt for brugeren at angive sine krav til studieplanen langt hurtigere, og det vil være muligt at præsentere den genererede studieplan visuelt i et skema.
En anden måde at gøre systemet mere brugervenligt på, kunne være ved at gøre det muligt for brugeren at oprette en profil i systemet. Dermed vil systemet kunne huske alle brugerens tidligere beståede kurser på baggrund af tidligere genererede studieplaner, og brugeren vil dermed kunne slippe for at angive sådanne informationer flere gange.
Det bedste måde at oprette et en form for profil-systemet, ville være hvis vores systemet blev integreret med DTUs CampusNet, hvor alle studerende i forvejen har en profil med tidligere valgte kurser. Med en sådan integration ville man også kunne tage højde for deltager-begrænsninger for kurser, således at systemet kan vælge de kurser fra, som der ikke er plads på.
En sådan integration vil dog være et omfattende projekt, og vil desuden kræve tilladelse og samarbejde med DTUs administration.

9.4 Foreslåede studieforløb

En anden funktionalitet der ville forbedre vores system, ville være at lave foreslåede studieforløb. Det vil sige at brugeren skulle kunne vælge at følge et foreslået studieforløb for en bestemt studieretning, hvorefter systemet sørger for at der bliver valgt kurser på en forudbestemt hensigtsmæssig måde. Eksempelvis kunne man sikre at en studieplan for Softwareteknologi altid har kurserne "Indledende programmering" og "Matematik 1" på første semester. Dette er også afspejlet i virkeligheden, da man som studerende automatisk bliver tilmeldt en række kurser når man starter på en bacheloruddannelse.

Konklusion

Overordnet set er det lykkedes os at lave et system til studieplanlægning, hvori en DTU studerende kan angive en række krav, og få genereret en studieplan der opfylder disse. Systemet opfylder de krav vi formulerede i vores kravspecifikation, og vi har ikke fundet nogen fejl i vores test af implementeringen. Dog er der forskellige fejl og uoverensstemmelser i den kursusdatabase vi benytter, hvilket betyder at der kan være fejl i de genererede studieplaner. Vores system har desuden to væsentlige problemer, som gør systemet upraktisk for studerende at bruge. Det ene problem er køretiden af systemet, da genereringen af studieplaner kan tage timevis afhængigt af studieplanens omfang og de angivne krav. Det andet problem er manglen på en brugervenlig grænseflade, således at brugeren nemmere kan angive sine krav, og nemmere kan overskue den genererede studieplan. Disse to problemer vil derfor være et oplagt sted at starte for en eventuel videreudvikling af vores system.

Appendiks A

Constraints

Indhold

A.1 Generelle studieplaner

```
/// No course overlap (S10)
let courseOverlapConstraint (courseVariables:IntExpr []) (courseDatabase:CourseDatabase)
    =
    let isCompoundScheduleCode arrayCelli =
        let compundScheduleCodes = [|E1; E2; E3; E4; E5; F1; F2; F3; F4; F5|]
        let boolArray =
            Array.map (fun sc ->
                ctx.MkEq(
                    ctx.MkMod(arrayCelli,ctx.MkInt numberOfScheduleCodes),
                    ctx.MkInt (codeToInt sc))) compundScheduleCodes
        ctx.MkOr(boolArray)

    let boolArray1 =
        [| for i in 0..((Array.length courseDatabase) - 1) ->
```

```
            let arrayCelli = courseVariables.[i]
            ctx.MkITE(
                ctx.MkGt(arrayCelli, ctx.MkInt 0), //if
                (let boolArray2 = //then
                    [| for j in 0..((Array.length courseDatabase) - 1) ->
                        let arrayCellj = courseVariables.[j]
                        ctx.MkITE(
                            ctx.MkAnd(ctx.MkGt(arrayCellj,ctx.MkInt 0), ctx.MkNot(ctx.
                                MkEq(ctx.MkInt i,ctx.MkInt j))), //if
                            ctx.MkITE(
                                isCompoundScheduleCode arrayCelli, //then if
                                ctx.MkAnd( //then
                                    ctx.MkNot(ctx.MkEq(arrayCelli, arrayCellj)),
                                    ctx.MkNot(ctx.MkEq(ctx.MkAdd(arrayCelli,ctx.MkInt
                                        1), arrayCellj)),
                                    ctx.MkNot(ctx.MkEq(ctx.MkAdd(arrayCelli,ctx.MkInt
                                        2), arrayCellj))),
                                ctx.MkNot(ctx.MkEq(arrayCelli, arrayCellj))) :?>
                                    BoolExpr, //else
                            ctx.MkBool true) :?> BoolExpr //else
                    |]
                 ctx.MkAnd(boolArray2)),
                ctx.MkBool true) :?> BoolExpr //else
        |]
    ctx.MkAnd(boolArray1)

/// No point blocked courses are taken (S11)
let pointBlock (courseVariables:IntExpr []) (courseDatabase:CourseDatabase) (
    courseToIndex:CourseToIndex) (passedCourses:PassedCourses)
            (courseNumberMap:CourseNumberMap) =
    let assertPointBlock (courseNumber:CourseNumber) =
        let index = Map.find courseNumber courseToIndex
        ctx.MkAnd(ctx.MkEq(courseVariables.[index], ctx.MkInt 0), ctx.MkNot(ctx.MkBool
            (Set.contains courseNumber passedCourses)))
    let boolArray =
        [| for i in 0..((Array.length courseDatabase) - 1) ->
            let (_,_,_,_,_,pb,_,_) = courseDatabase.[i]
            let pbAlt =
                Set.fold (fun acc cn ->
                    let set = Map.tryFind cn courseNumberMap
                    match set with
                    | Some s -> Set.union s acc
                    | _ -> acc) Set.empty pb
            let boolArray = Set.toArray (Set.map (fun x -> assertPointBlock x) pbAlt)
            ctx.MkITE(ctx.MkGt(courseVariables.[i], ctx.MkInt 0), ctx.MkAnd(boolArray),
                ctx.MkBool true) :?> BoolExpr
        |]
    ctx.MkAnd(boolArray)

/// Course prerequisites (S12)
let coursePrerequisites (courseVariables:IntExpr []) (courseDatabase:CourseDatabase) (
    courseToIndex:CourseToIndex)
                    (passedCourses:PassedCourses) (courseNumberMap:
                        CourseNumberMap) =
```

```
    // We create a new set consisting of BoolExpr where the value of each element
        coresponds to whether the given course is taken
    // on a previous period, where a period is either a 13-week period or a 3-week
        period.
    let coursePrerequisites'' (cpd:CoursePrerequisitesDisjunctions) (scheduleCodeInt:
        IntExpr) =
        let boolSet =
            Set.map (fun cn ->
                ctx.MkOr(
                    ctx.MkBool(Set.contains cn passedCourses),
                    ctx.MkAnd(
                        ctx.MkGt(
                            courseVariables.[Map.find cn courseToIndex], ctx.MkInt 0),
                            ctx.MkITE(
                                ctx.MkEq(ctx.MkMod(scheduleCodeInt, ctx.MkInt ((
                                    numberOfScheduleCodes / 2) - 1)), ctx.MkInt 0), //
                                    if three weeks course
                                ctx.MkLt(courseVariables.[Map.find cn courseToIndex],
                                    scheduleCodeInt), //then
                                ctx.MkLe(courseVariables.[Map.find cn courseToIndex], //
                                    else
                                    // Last schedule code on previous semester
                                    ctx.MkMul(
                                        ctx.MkDiv(scheduleCodeInt, ctx.MkInt (
                                            numberOfScheduleCodes / 2)),
                                        ctx.MkInt (numberOfScheduleCodes / 2)))) :?>
                                            BoolExpr))) cpd
        let boolArray = Set.toArray boolSet
        ctx.MkOr(boolArray)

    // We check whether the prerequisitets for a given course are met.
    let coursePrerequisites' (cp:CoursePrerequisites) (scheduleCodeInt:IntExpr) =
        let boolSet =
            Set.map (fun cpd ->
                let cpdAlt =
                    Set.fold (fun acc cn ->
                        let set = Map.tryFind cn courseNumberMap
                        match set with
                        | Some cns -> Set.union cns acc
                        | _ -> acc) Set.empty cpd
                coursePrerequisites'' cpdAlt scheduleCodeInt) cp
        let boolArray = Set.toArray boolSet
        ctx.MkAnd(boolArray)

    // For all chosen courses we call the function above appropriately
    let boolArray =
        [| for i in 0..((Array.length courseDatabase) - 1) ->
            let (cn,_,_,_,_,_,mcp,_) = courseDatabase.[i]
            let arrayCell = courseVariables.[i]
            ctx.MkImplies(ctx.MkGt(arrayCell, ctx.MkInt 0), coursePrerequisites' mcp
                arrayCell)
        |]
    ctx.MkAnd(boolArray)

/// The studyplan has the right amount of semesters (OI1)
```

```
let rightAmountOfSemesters (courseVariables:IntExpr []) (studyLength:StudyLength) (
    semesterStartInSpring:SemesterStartInSpring) =
    let boolArray =
        [| for c in courseVariables ->
            if semesterStartInSpring
            then
                ctx.MkOr(
                    ctx.MkEq(c, ctx.MkInt 0),
                    // If the first semester is in the spring the studyplan is shifted
                        forward one semester
                    ctx.MkAnd(
                        ctx.MkLe(c, ctx.MkInt ((numberOfScheduleCodes / 2) * (
                            studyLength + 1))),
                        ctx.MkGe(c, ctx.MkInt ((numberOfScheduleCodes / 2) + 1))))
            else ctx.MkAnd(ctx.MkLe(c, ctx.MkInt ((numberOfScheduleCodes / 2) *
                studyLength)), ctx.MkGe(c, ctx.MkInt 0))
        |]
    ctx.MkAnd(boolArray)

/// No courses can be chosen in a study abroad semester (VI1)
let studyAbroadConstraint (courseVariables:IntExpr []) (sas:Set<StudyAbroadSemester>)
    =
    let studyAbroadConstraint' (semester:int) =
        // First schedule code in the given semester
        let min = (semester - 1) * (numberOfScheduleCodes / 2) + 1
        // Last schedule code in the given semester
        let max = semester * (numberOfScheduleCodes / 2)
        let boolArray =
            [| for c in courseVariables ->
                ctx.MkNot(ctx.MkAnd(ctx.MkGe(c, ctx.MkInt min),ctx.MkLe(c, ctx.MkInt
                    max)))
            |]
        ctx.MkAnd(boolArray)
    let boolArray = Set.toArray (Set.map (fun (_,s) -> studyAbroadConstraint' s) sas)
    ctx.MkAnd(boolArray)

/// Passed courses can not be chosen (VI2)
let passedCoursesConstraint (courseVariables:IntExpr []) (passedCourses:PassedCourses) (
    courseDatabase:CourseDatabase)
                            (courseToIndex:CourseToIndex) =
    let boolArray = Set.toArray (Set.map (fun c -> ctx.MkEq(courseVariables.[Map.find c
        courseToIndex], ctx.MkInt 0)) passedCourses)
    ctx.MkAnd(boolArray)

/// Constraints for 3-week period schedule codes that must be occupied (VK2)
let threeWeekPeriodConstraint (courseVariables:IntExpr []) (twpcs:
    ThreeWeekPeriodConstraints) =
    let threeWeekPeriodHardConstraints' (scheduleCodeInteger:int) =
        let boolArray =
            [| for i in 0..(( Array.length courseVariables) - 1) ->
                ctx.MkEq(courseVariables.[i], ctx.MkInt scheduleCodeInteger)
            |]
        ctx.MkOr(boolArray)
    let boolArray =
        [| for i in 1..( Array.length twpcs) ->
```

```
            if twpcs.[i - 1]
            then
                let scheduleCodeInteger = (i * (numberOfScheduleCodes / 2)) - 1
                threeWeekPeriodHardConstraints' scheduleCodeInteger
            else ctx.MkBool true
        |]
    ctx.MkAnd(boolArray)

/// Constraint: ECTS points for all semesters match the assigned amount (VK3)
let semesterPointConstraints (courseVariables:IntExpr []) courseVariablesLength (
     ectsPoints:IntNum [])
                            (ectsDistribution:ECTSDistribution) (studyAbroadSemesters:
                                StudyAbroadSemesters) =
    let boolArray =
        [| for i in 0..((Array.length ectsDistribution) - 1) ->
            let lowerBound = ((numberOfScheduleCodes / 2) * i) + 1
            let upperBound = (numberOfScheduleCodes / 2) * (i + 1)
            let sumArray =
                [| for j in 0..(courseVariablesLength - 1) ->
                    let arrayCell = courseVariables.[j]
                    ctx.MkITE(
                        ctx.MkAnd( //if
                            ctx.MkGe(arrayCell, ctx.MkInt lowerBound),
                            ctx.MkLe(arrayCell, ctx.MkInt upperBound)),
                        ectsPoints.[j], //then
                        ctx.MkInt 0) :?> ArithExpr //else
                |]
            let sum = ctx.MkAdd(sumArray)
            let sumAbroad = int ((Set.fold (fun acc (p,s) -> if s = i + 1 then p + acc
                 else acc + 0.0) 0.0 studyAbroadSemesters) * 10.0)
            let (min, max) = ectsDistribution.[i]
            ctx.MkAnd(
                ctx.MkGe(ctx.MkAdd(sum, ctx.MkInt sumAbroad), ctx.MkInt (int (min *
                    10.0))),
                ctx.MkLe(ctx.MkAdd(sum, ctx.MkInt sumAbroad), ctx.MkInt (int (max *
                    10.0))))
        |]
    ctx.MkAnd(boolArray)

/// Constraints for schedule codes that must be free (VK4)
let freeScheduleCodes (courseVariables:IntExpr []) (scuhcs:
     ScheduleCodeUnwantedHardConstraints) =
    let freeScheduleCodes'' (scheduleCodeInt:int) =
        let boolArray =
            [| for c in courseVariables ->
                ctx.MkNot(ctx.MkEq(c, ctx.MkInt scheduleCodeInt))
            |]
        ctx.MkAnd(boolArray)

    let freeScheduleCodes' (scc:ScheduleCodeConstraint) =
        let (sc,s) = scc
        let year = (s + 1) / 2
        let sci = codeToInt sc
        let boolArray =
            match sc with
```

```
            | E1 | E2 | E3 | E4 | E5 | F1 | F2 | F3 | F4 | F5 ->
                [| for i in 0..2 -> freeScheduleCodes'' (sci + ((year - 1) *
                    numberOfScheduleCodes) + i) |]
            | E1A | E2A | E3A | E4A | E5A | F1A | F2A | F3A | F4A | F5A ->
                [| for i in 0..1 -> freeScheduleCodes'' (sci + ((year - 1) *
                    numberOfScheduleCodes) - i) |]
            | E1B | E2B | E3B | E4B | E5B | F1B | F2B | F3B | F4B | F5B ->
                [| freeScheduleCodes'' (sci + ((year - 1) * numberOfScheduleCodes));
                   freeScheduleCodes'' (sci + ((year - 1) * numberOfScheduleCodes) - 2)
                        |]
            | E | F -> [|freeScheduleCodes'' (sci + ((year - 1) * numberOfScheduleCodes
                ))|]
            | _ -> [|ctx.MkBool true|]
        ctx.MkAnd(boolArray)

    let boolArray = Set.toArray (Set.map (fun scc -> freeScheduleCodes' scc) scuhcs)
    ctx.MkAnd(boolArray)

/// Constraints for courses that must be chosen (VK7)
let selectedCourses (courseVariables:IntExpr []) (cwhcs:CourseWantedHardConstraints) (
    courseToIndex:CourseToIndex)
                (courseNumberMap:CourseNumberMap) =
    let selectedCourses' (cwhc:CourseWantedHardConstraint) =
        let selectedCourses'' (cwhc:CourseWantedHardConstraint) =
            let (cn,semester) = cwhc
            match semester with
            // If no requirement for the semester, the course needs only to be chosen
            | 0 -> ctx.MkGt(courseVariables.[Map.find cn courseToIndex], ctx.MkInt 0)
            // Else it needs to be chosen between the first schedule code in the given
                semester and the last
            | _ -> ctx.MkAnd(
                        ctx.MkGt(
                            courseVariables.[Map.find cn courseToIndex],
                            ctx.MkInt ((semester - 1) * (numberOfScheduleCodes / 2))),
                        ctx.MkLe(
                            courseVariables.[Map.find cn courseToIndex],
                            ctx.MkInt (semester * (numberOfScheduleCodes / 2))))

        let (cn,semester) = cwhc
        let cwhcAlt = Map.tryFind cn courseNumberMap
        match cwhcAlt with
        | Some cns -> let set = Set.map (fun cn' -> (cn',semester)) cns
                      let boolArray = Set.toArray (Set.map (fun cwhc -> selectedCourses
                          '' cwhc) set)
                      ctx.MkOr(boolArray)
        | _ -> failwith "Incorrect course number in wanted courses"

    let boolArray = Set.toArray (Set.map (fun cwhc -> selectedCourses' cwhc) cwhcs)
    ctx.MkAnd(boolArray)

/// Constraints for courses that must not be chosen (VK9)
let deselectedCourses (courseVariables:IntExpr []) (cuhcs:
    CourseUnwantedHardConstraints) (courseToIndex:CourseToIndex)
                (courseNumberMap:CourseNumberMap) =
    let deselectedCourses' (cuhc:CourseUnwantedHardConstraint) =
```

```
        let (cn,semester) = cuhc
        match semester with
        // If no requirement for the semester, the course can not be chosen at all
        | 0 -> ctx.MkEq(courseVariables.[Map.find cn courseToIndex], ctx.MkInt 0)
        // Else it needs only to be outside of the interval that is between the first
             schedule code in the given semester and the last
        | _ -> ctx.MkNot(
                    ctx.MkAnd(
                        ctx.MkGt(
                            courseVariables.[Map.find cn courseToIndex],
                            ctx.MkInt ((semester - 1) * (numberOfScheduleCodes / 2))),
                        ctx.MkLe(
                            courseVariables.[Map.find cn courseToIndex],
                            ctx.MkInt (semester * (numberOfScheduleCodes / 2)))))
    let cuhcsAlt =
        Set.fold (fun acc (cn,s) ->
            let set = Map.tryFind cn courseNumberMap
            match set with
            | Some cns ->
                let set' = Set.map (fun cn' -> (cn',s)) cns
                Set.union set' acc
            | _ -> acc) Set.empty cuhcs
    let boolArray = Set.toArray (Set.map (fun cuhc -> deselectedCourses' cuhc) cuhcsAlt
        )
    ctx.MkAnd(boolArray)

/// All fields have the required ECTS point (VK10)
let fieldRequirementsConstraint (courseVariables:IntExpr []) (courseDatabase:
    CourseDatabase) (fieldRequirements:FieldRequirements) =
    let fieldRequirementsConstraint' (fieldRequirement:FieldRequirement) =
        let (field,sumreq) = fieldRequirement
        let sumArray =
            [|for i in 0..((Array.length courseDatabase) - 1) ->
                let (_,_,ECTS,_,_,_,_,courseFields) = courseDatabase.[i]
                if Set.contains field courseFields
                then
                    let arrayCell = courseVariables.[i]
                    ctx.MkITE(
                        ctx.MkGt(arrayCell,ctx.MkInt 0), //if
                        ctx.MkInt (int (ECTS * 10.0)), //then
                        ctx.MkInt 0) :?> ArithExpr //else
                else (ctx.MkInt 0) :> ArithExpr
            |]
        let sum = ctx.MkAdd(sumArray)
        ctx.MkGe(sum,ctx.MkInt (int (sumreq * 10.0)))

    let boolArray =
        [|for fr in fieldRequirements ->
            fieldRequirementsConstraint' fr
        |]
    ctx.MkAnd(boolArray)
```

A.2 Bachelor-studieplaner

```
/// The study plan consists of courses corresponding to 180 ECTS point (S1)
let bachelorPointConstraint (courseVariables:IntExpr []) (ectsPoints:IntNum []) (
    passedCourses:PassedCourses) (courseDatabase:CourseDatabase)
                            (studyAbroadSemesters:StudyAbroadSemesters) =
    let sumArray =
        [| for i in 0..((Array.length courseVariables) - 1) ->
            let (cn,_,_,_,_,_,_,_) = courseDatabase.[i]
            ctx.MkITE(
                ctx.MkOr(
                    ctx.MkGt(courseVariables.[i], ctx.MkInt 0), //if
                    ctx.MkBool (Set.contains cn passedCourses)),
                ectsPoints.[i], //then
                ctx.MkInt 0) :?> ArithExpr //else
        |]
    let sum = ctx.MkAdd(sumArray)
    let abroadSum = int ((Set.fold (fun acc (p,_) -> p + acc) 0.0 studyAbroadSemesters)
        * 10.0)
    ctx.MkEq(ctx.MkAdd(sum, ctx.MkInt abroadSum), ctx.MkInt (180 * 10))

/// A bachelor can have no more than 12 semesters (S2)
let bachelorLengthConstraint (studyLength:StudyLength) =
    ctx.MkLe(ctx.MkInt studyLength, ctx.MkInt 12)

/// All mandatory courses for a given bachelor study are chosen (S4)
let mandatoryBachelorCourses (courseVariables:IntExpr []) (study:BachelorStudy) (
    courseToIndex:CourseToIndex) (courseNumberMap:CourseNumberMap)
                            (passedCourses : PassedCourses) =
    let (_,mandatoryStudyCourses,_,_,_,_,_,_) = study
    let boolArray =
        Set.toArray (Set.map (fun msc ->
                        let mscAlts =
                            let set = Map.tryFind msc courseNumberMap
                            match set with
                            | Some cn -> cn
                            | _ -> Set.empty
                        let boolArray =
                            Set.toArray(
                                Set.map (fun mscAlt ->
                                    ctx.MkOr(
                                        ctx.MkGt(
                                            courseVariables.[Map.find mscAlt
                                                courseToIndex],
                                            ctx.MkInt 0),
                                        ctx.MkBool (Set.contains mscAlt passedCourses)))
                                            mscAlts)
                        ctx.MkOr(boolArray)) mandatoryStudyCourses)
    ctx.MkAnd(boolArray)

/// No phd courses on bachelor (S6)
let bachelorCourseLevelConstraint (courseVariables:IntExpr []) (courseDatabase:
    CourseDatabase) =
    let boolArray =
```

```
        [| for i in 0..((Array.length courseVariables) - 1) ->
            let (_,_,_,level,_,_,_,_) = courseDatabase.[i]
            ctx.MkImplies(
                ctx.MkGt(courseVariables.[i], ctx.MkInt 0),
                ctx.MkLt(ctx.MkInt level, ctx.MkInt 3))
        |]
    ctx.MkAnd(boolArray)

/// All course types have a minimum of 45 ECTS points (S7)
let courseTypeConstraint (courseVariables:IntExpr []) (courseDatabase:CourseDatabase) (
     study:BachelorStudy) (courseNumberMap:CourseNumberMap)
                          (passedCourses:PassedCourses) =
    let courseTypeConstraint' (courseType:Set<CourseNumber>) (point:int) =
        let courseTypeAlt =
            Set.fold (fun acc cn ->
                let set = Map.tryFind cn courseNumberMap
                match set with
                | Some cns -> Set.union cns acc
                | _ -> acc) Set.empty courseType
        let sumArray =
            [| for i in 0..((Array.length courseVariables) - 1) ->
                let (cn,_,ects,_,_,_,_,_) = courseDatabase.[i]
                ctx.MkITE(
                    ctx.MkAnd( //if
                        ctx.MkOr(
                            ctx.MkGt(courseVariables.[i], ctx.MkInt 0),
                            ctx.MkBool (Set.contains cn passedCourses)),
                        ctx.MkBool(Set.contains cn courseTypeAlt)),
                    ctx.MkInt (int (ects * 10.0)), //then
                    ctx.MkInt 0) :?> ArithExpr //else
            |]

        let sum = ctx.MkAdd(sumArray)
        ctx.MkGe(sum, ctx.MkInt (point * 10))

    let (_,_,basicScienceCourses,technologicalStudyCourses,projectsAndCommonCourses)
      = study
    let basicScienceCoursesConstraint = courseTypeConstraint' basicScienceCourses 45
    let technologicalStudyCoursesConstraint = courseTypeConstraint'
      technologicalStudyCourses 45
    let projectsAndCommonCoursesConstraint = courseTypeConstraint'
      projectsAndCommonCourses 30 // Bachelor project excluded
    ctx.MkAnd(basicScienceCoursesConstraint,technologicalStudyCoursesConstraint,
      projectsAndCommonCoursesConstraint)

/// Bachelor project must be chosen after at least 90 ECTS point (S13)
let bachelorProjectConstraint (courseVariables:IntExpr []) (courseToIndex:CourseToIndex)
    (ectsPoints:IntNum [])
                              (passedCourses:PassedCourses) (courseDatabase:
                                  CourseDatabase) =
    let point = 90
    let calculateSumBefore (index : int) =
        let sumArray =
            [| for i in 0..((Array.length courseVariables) - 1) ->
                let (cn,_,_,_,_,_,_,_) = courseDatabase.[i]
```

```
                ctx.MkITE(
                    ctx.MkOr(
                        ctx.MkAnd(
                            ctx.MkGt(courseVariables.[i], ctx.MkInt 0),
                            ctx.MkLe(
                                courseVariables.[i],
                                // The last schedule code on the previous semester:
                                // (code - 1) - ((code - 1) % (numberOfScheduleCode /
                                    2))
                                ctx.MkSub(
                                    ctx.MkSub(courseVariables.[index], ctx.MkInt 1),
                                    ctx.MkMod(
                                        ctx.MkSub(courseVariables.[index], ctx.MkInt 1)
                                            :?> IntExpr,
                                        ctx.MkInt (numberOfScheduleCodes / 2))))),
                        ctx.MkBool(Set.contains cn passedCourses)),
                    ectsPoints.[i], //then
                    ctx.MkInt 0) :?> ArithExpr //else
            |]
        ctx.MkAdd(sumArray)

    let index15E = Map.find "Bac15E" courseToIndex
    let index15F = Map.find "Bac15F" courseToIndex
    let index20E = Map.find "Bac20E" courseToIndex
    let index20F = Map.find "Bac20F" courseToIndex
    let bac15Bool =
        ctx.MkAnd(
            ctx.MkITE(
                ctx.MkGt(courseVariables.[index15E], ctx.MkInt 0), //if
                ctx.MkGe(calculateSumBefore index15E, ctx.MkInt (point * 10)), //then
                ctx.MkBool true) :?> BoolExpr, //else
            ctx.MkITE(
                ctx.MkGt(courseVariables.[index15F], ctx.MkInt 0), //if
                ctx.MkGe(calculateSumBefore index15F, ctx.MkInt (point * 10)), //then
                ctx.MkBool true) :?> BoolExpr) //else

    let bac20Bool =
        ctx.MkAnd(
            ctx.MkITE(
                ctx.MkGt(courseVariables.[index20E], ctx.MkInt 0),
                ctx.MkGe(calculateSumBefore index20E, ctx.MkInt (point * 10)),
                ctx.MkBool true) :?> BoolExpr,
            ctx.MkITE(
                ctx.MkGt(courseVariables.[index20F], ctx.MkInt 0),
                ctx.MkGe(calculateSumBefore index20F, ctx.MkInt (point * 10)),
                ctx.MkBool true) :?> BoolExpr)

    let pointBool = ctx.MkAnd(bac15Bool, bac20Bool)
    let chosenBool =
        ctx.MkOr(
            ctx.MkGt(courseVariables.[index15E], ctx.MkInt 0),
            ctx.MkGt(courseVariables.[index15F], ctx.MkInt 0),
            ctx.MkGt(courseVariables.[index20E], ctx.MkInt 0),
            ctx.MkGt(courseVariables.[index20F], ctx.MkInt 0))
    ctx.MkAnd(pointBool, chosenBool)
```

A.3 Kandidat-studieplaner

```
/// The study plan consists of courses corresponding to 120 ECTS point (S1)
let masterPointConstraint (courseVariables:IntExpr []) (ectsPoints:IntNum []) (
    passedMasterCourses:PassedCourses)
                          (courseDatabase:CourseDatabase) (studyAbroadSemesters:
                              StudyAbroadSemesters) =
    let sumArray =
        [| for i in 0..((Array.length courseVariables) - 1) ->
            let (cn,_,_,_,_,_,_,_) = courseDatabase.[i]
            ctx.MkITE(
                ctx.MkOr( //if
                    ctx.MkGt(courseVariables.[i], ctx.MkInt 0),
                    ctx.MkBool(Set.contains cn passedMasterCourses)),
                ectsPoints.[i], //then
                ctx.MkInt 0) :?> ArithExpr //else
            |]
    let sum = ctx.MkAdd(sumArray)
    let abroadSum = int ((Set.fold (fun acc (p,_) -> p + acc) 0.0 studyAbroadSemesters)
        * 10.0)
    ctx.MkEq(ctx.MkAdd(sum, ctx.MkInt abroadSum), ctx.MkInt (120 * 10))

/// A master can have no more than 8 semesters (S2)
let masterLengthConstraint (studyLength:StudyLength) =
    ctx.MkLe(ctx.MkInt studyLength, ctx.MkInt 8)

/// Basic courses on master accounts for no more than 10 ECTS points (S5)
let masterCourseLevelConstraint (courseVariables:IntExpr []) (courseDatabase:
    CourseDatabase) =
    let sumArray =
        [| for i in 0..((Array.length courseVariables) - 1) ->
            let (_,_,ects,level,_,_,_,_) = courseDatabase.[i]
            ctx.MkITE(
                ctx.MkAnd(
                    ctx.MkGt(courseVariables.[i], ctx.MkInt 0),
                    ctx.MkEq(ctx.MkInt level, ctx.MkInt 1)), //if
                ctx.MkInt (int (ects * 10.0)), //then
                ctx.MkInt 0) :?> ArithExpr //else
        |]
    let sum = ctx.MkAdd(sumArray)
    ctx.MkLe(sum, ctx.MkInt 100)


let ComputerScienceAndEngineeringConstraints (courseVariables:IntExpr []) (
    courseDatabase:CourseDatabase) (masterStudy:MasterStudy)
                                                (ctx:Context) (courseToIndex:CourseToIndex
                                                    ) (courseNumberMap:
                                                    CourseNumberMap)
                                                (passedCourses : PassedCourses) =
    let StudyTypeSum (courseVariables:IntExpr []) (courseDatabase:CourseDatabase) (
        studyCourses:Set<CourseNumber>) (ctx:Context) =
        let sumArray =
            [| for i in 0..((Array.length courseVariables) - 1) ->
                let (cn,_,ects,_,_,_,_,_) = courseDatabase.[i]
                ctx.MkITE(
```

```
                    ctx.MkAnd(
                        ctx.MkOr(
                            ctx.MkGt(courseVariables.[i], ctx.MkInt 0),
                            ctx.MkBool (Set.contains cn passedCourses)),
                        ctx.MkEq(ctx.MkBool (Set.contains cn studyCourses), ctx.MkBool
                             true)), //if
                        ctx.MkInt (int (ects * 10.0)), //then
                        ctx.MkInt 0) :?> ArithExpr //else
            |]
        ctx.MkAdd(sumArray)

    let (_,gcc,tsc) = masterStudy
    let gccAlt =
        Set.fold (fun acc cn ->
            let set = Map.tryFind cn courseNumberMap
            match set with
            | Some cns -> Set.union cns acc
            | _ -> acc) Set.empty gcc
    let tscAlt =
        Set.fold (fun acc cn ->
            let set = Map.tryFind cn courseNumberMap
            match set with
            | Some cns -> Set.union cns acc
            | _ -> acc) Set.empty tsc

    // The point sum of chosen General Competence Courses
    let sumgcc = StudyTypeSum courseVariables courseDatabase gccAlt ctx
    let gccSumBool =
        ctx.MkITE(
            ctx.MkGt(courseVariables.[Map.find "42435" courseToIndex], ctx.MkInt 0), //if
            ctx.MkGe(sumgcc, ctx.MkInt 250), //then
            ctx.MkGe(sumgcc, ctx.MkInt 200)) :?> BoolExpr //else
    // The point sum of chosen Technological Specialization Courses
    let sumtsc = StudyTypeSum courseVariables courseDatabase tscAlt ctx
    let tscSumBool = ctx.MkGe(sumtsc, ctx.MkAdd(sumgcc, ctx.MkInt 300)) //Overlap
         between gcc and tsc
    let mandatoryCourseBool =
        ctx.MkOr(
            ctx.MkGt(courseVariables.[Map.find "42435" courseToIndex], ctx.MkInt 0),
            ctx.MkGt(courseVariables.[Map.find "42490A" courseToIndex], ctx.MkInt 0),
            ctx.MkGt(courseVariables.[Map.find "42490B" courseToIndex], ctx.MkInt 0),
            ctx.MkBool (Set.contains "42435" passedCourses),
            ctx.MkBool (Set.contains "42490A" passedCourses),
            ctx.MkBool (Set.contains "42490B" passedCourses))
    ctx.MkAnd(tscSumBool,gccSumBool,mandatoryCourseBool)

/// Course distribution for master study (S8)
let masterStudyCourseDistribution (courseVariables:IntExpr []) (courseDatabase:
     CourseDatabase) (studyName:StudyName)
                                  (masterStudy:MasterStudy) (courseToIndex:
                                       CourseToIndex) (courseNumberMap:
                                       CourseNumberMap) =
    match studyName with
    | "Informationsteknologi" -> ComputerScienceAndEngineeringConstraints
        courseVariables courseDatabase masterStudy ctx courseToIndex
```

```
            courseNumberMap
    | _ -> failwith "Not a valid master study"

/// Master thesis must be chosen after at least 55 ECTS point S(13)
let masterThesisConstraint (courseVariables:IntExpr []) (courseToIndex:CourseToIndex) (
    ectsPoints:IntNum [])
                                (passedMasterCourses:PassedCourses) (courseDatabase:
                                    CourseDatabase) =
    let point = 55
    let calculateSumBefore (index : int) =
        let sumArray =
            [| for i in 0..((Array.length courseVariables) - 1) ->
                let (cn,_,_,_,_,_,_,_) = courseDatabase.[i]
                ctx.MkITE(
                    ctx.MkOr(
                        ctx.MkAnd(
                            ctx.MkGt(courseVariables.[i], ctx.MkInt 0),
                            ctx.MkLe(
                                courseVariables.[i],
                                // The last schedule code on the previous semester:
                                // (code - 1) - ((code - 1) % (numberOfScheduleCode /
                                    2))
                                ctx.MkSub(
                                    ctx.MkSub(courseVariables.[index], ctx.MkInt 1),
                                    ctx.MkMod(
                                        ctx.MkSub(courseVariables.[index], ctx.MkInt 1)
                                            :?> IntExpr,
                                        ctx.MkInt (numberOfScheduleCodes / 2))))),
                        ctx.MkBool(Set.contains cn passedMasterCourses)),
                    ectsPoints.[i], //then
                    ctx.MkInt 0) :?> ArithExpr //else
            |]
        ctx.MkAdd(sumArray)

    let index30E = Map.find "Mas30E" courseToIndex
    let index30F = Map.find "Mas30F" courseToIndex
    let index32E = Map.find "Mas32E" courseToIndex
    let index32F = Map.find "Mas32F" courseToIndex
    let index35E = Map.find "Mas35E" courseToIndex
    let index35F = Map.find "Mas35F" courseToIndex

    let mas30Bool =
        ctx.MkAnd(
            ctx.MkITE(
                ctx.MkGt(courseVariables.[index30E], ctx.MkInt 0), //if
                ctx.MkGe(calculateSumBefore index30E, ctx.MkInt (point * 10)), //then
                ctx.MkBool true) :?> BoolExpr, //else
            ctx.MkITE(
                ctx.MkGt(courseVariables.[index30F], ctx.MkInt 0),
                ctx.MkGe(calculateSumBefore index30F, ctx.MkInt (point * 10)),
                ctx.MkBool true) :?> BoolExpr)

    let mas32Bool =
        ctx.MkAnd(
            ctx.MkITE(
```

```
                ctx.MkGt(courseVariables.[index32E], ctx.MkInt 0),
                ctx.MkGe(calculateSumBefore index32E, ctx.MkInt (point * 10)),
                ctx.MkBool true) :?> BoolExpr,
            ctx.MkITE(
                ctx.MkGt(courseVariables.[index32F], ctx.MkInt 0),
                ctx.MkGe(calculateSumBefore index32F, ctx.MkInt (point * 10)),
                ctx.MkBool true) :?> BoolExpr)

    let mas35Bool =
        ctx.MkAnd(
            ctx.MkITE(
                ctx.MkGt(courseVariables.[index35E], ctx.MkInt 0),
                ctx.MkGe(calculateSumBefore index35E, ctx.MkInt (point * 10)),
                ctx.MkBool true) :?> BoolExpr,
            ctx.MkITE(
                ctx.MkGt(courseVariables.[index35F], ctx.MkInt 0),
                ctx.MkGe(calculateSumBefore index35F, ctx.MkInt (point * 10)),
                ctx.MkBool true) :?> BoolExpr)

    let pointBool = ctx.MkAnd(mas30Bool, mas32Bool, mas35Bool)
    let chosenBool =
        ctx.MkOr(
            ctx.MkGt(courseVariables.[index30E], ctx.MkInt 0),
            ctx.MkGt(courseVariables.[index30F], ctx.MkInt 0),
            ctx.MkGt(courseVariables.[index32E], ctx.MkInt 0),
            ctx.MkGt(courseVariables.[index32F], ctx.MkInt 0),
            ctx.MkGt(courseVariables.[index35E], ctx.MkInt 0),
            ctx.MkGt(courseVariables.[index35F], ctx.MkInt 0))
    ctx.MkAnd(pointBool, chosenBool)
```

A.4 Øvrige constraints

```
/// Split courses are chosen correctly
let partialCourseConstraint (courseVariables:IntExpr []) (pcpss:Set<PartialCoursePairs>)
  (courseDatabase:CourseDatabase) =
    let partialCourseConstraint' (pcps:PartialCoursePairs) =
        let partialCourseConstraint'' (pcp:int * ScheduleCode) (pcps:Set<int *
          ScheduleCode>) =
            let partialCourseConstraint''' (pcp1:int * ScheduleCode) (pcp2:int *
              ScheduleCode) =
                let (index1, scheduleCode1) = pcp1
                let (index2, scheduleCode2) = pcp2

                // Both partial courses are chosen in the correct order
                let scheduleCode1Int = codeToInt scheduleCode1
                let scheduleCode2Int = codeToInt scheduleCode2
                let difference = scheduleCode2Int - scheduleCode1Int
                ctx.MkImplies(
                    ctx.MkOr(
                        ctx.MkGt(courseVariables.[index1], ctx.MkInt 0),
                        ctx.MkGt(courseVariables.[index2], ctx.MkInt 0)),
                    ctx.MkEq(
                        courseVariables.[index2],
```

```
                    ctx.MkAdd(courseVariables.[index1],ctx.MkInt difference)))

            let boolArray = Set.toArray (Set.map (fun pcp2 -> partialCourseConstraint'''
                pcp pcp2) pcps)
            ctx.MkAnd(boolArray)

        let boolArray = Set.toArray (Set.map (fun pcp -> partialCourseConstraint'' pcp
            (Set.remove pcp pcps)) pcps)
        ctx.MkAnd(boolArray)

    let boolArray = Set.toArray (Set.map (fun pcps -> partialCourseConstraint' pcps)
        pcpss)
    ctx.MkAnd(boolArray)

/// The courses are assigned to a correct schedule code
let sameScheduleCodeConstraint (courseVariables:IntExpr []) (courseDatabase:
    CourseDatabase) =
    let boolArray =
        [| for i in 0..((Array.length courseDatabase) - 1) ->
            let arrayCell = courseVariables.[i]
            let s1 =
                ctx.MkITE(
                    ctx.MkLe(arrayCell,ctx.MkInt numberOfScheduleCodes), //if
                    arrayCell, //then
                    ctx.MkAdd(
                        ctx.MkMod(
                            ctx.MkSub(arrayCell, ctx.MkInt 1) :?> IntExpr,
                            ctx.MkInt numberOfScheduleCodes), ctx.MkInt 1)) //else
            let (_,_,_,_,scheduleCode,_,_,_) = courseDatabase.[i]
            let s2 = ctx.MkInt (codeToInt scheduleCode)
            ctx.MkImplies(ctx.MkGt(arrayCell, ctx.MkInt 0), ctx.MkEq(s1, s2))
                |]
    ctx.MkAnd(boolArray)

/// A partial studyplan must be expandable to a valid studyplan (S14)
let expandableToValid (courseVariables:IntExpr []) (studyLevel:StudyLevel) (
    inputCourseDatabase:InputCourseDatabase)
                    (courseConstraints:CourseConstraints) (scheduleCodeConstraints:
                        ScheduleCodeConstraints) (ectsDistribution:ECTSDistribution)
                    (threeWeekPeriodHardConstraints:ThreeWeekPeriodConstraints) (
                        studyLength:StudyLength)
                    (fieldRequirements:FieldRequirements) (studyName:StudyName) (
                        passedCourses:PassedCourses)
                    (passedMasterCourses:PassedCourses) (semesterStartInSpring:
                        SemesterStartInSpring)
                    (courseNumberMap : CourseNumberMap) (studyAbroadSemesters:
                        StudyAbroadSemesters) =

    // (CourseUnwantedHardConstraints, CourseWantedHardConstraints,
        CourseWantedSoftConstraints)
    let (cwscs, cwhcs, cuhcs) = courseConstraints

    let solution =
        if studyLevel = 1
        then
```

```
            let extendedECTSDistribution =
                [|for i in 0..11 ->
                    if i < Array.length ectsDistribution
                    then ectsDistribution.[i]
                    else (0.0,40.0)
                |]

            let cuhcsNew =
                Set.fold (fun acc c -> let (sc,s) = c
                                       if s = 0
                                       then
                                           let constraintList = [for i in (studyLength +
                                               1)..12 -> (sc,i)]
                                           let constraintSet = set constraintList
                                           Set.union acc constraintSet
                                       else acc
                         )(set []) cwhcs
            let cuhcs = Set.union cuhcs cuhcsNew

            solveValidBachelor inputCourseDatabase (cwscs, cwhcs, cuhcs)
                scheduleCodeConstraints extendedECTSDistribution
                    threeWeekPeriodHardConstraints 12 fieldRequirements
                        studyName
                    passedCourses semesterStartInSpring courseNumberMap
                        studyAbroadSemesters

        else
            let extendedECTSDistribution =
                [|for i in 0..7 ->
                    if i < Array.length ectsDistribution
                    then ectsDistribution.[i]
                    else (0.0,40.0)
                |]

            let cuhcsNew =
                Set.fold (fun acc c -> let (sc,s) = c
                                       if s = 0
                                       then
                                           let constraintList = [for i in (studyLength +
                                               1)..8 -> (sc,i)]
                                           let constraintSet = set constraintList
                                           Set.union acc constraintSet
                                       else acc
                         )(set []) cwhcs
            let cuhcs = Set.union cuhcs cuhcsNew

            solveValidMaster inputCourseDatabase (cwscs, cwhcs, cuhcs)
                scheduleCodeConstraints extendedECTSDistribution
                    threeWeekPeriodHardConstraints 8 fieldRequirements
                        studyName
                    passedCourses passedMasterCourses semesterStartInSpring
                        courseNumberMap studyAbroadSemesters

    match solution with
    | None -> ctx.MkBool false
```

```
| Some (_,m) ->
    let maxScheduleCode = (numberOfScheduleCodes / 2) * studyLength
    let boolArray =
        [| for i in 0..((Array.length courseVariables) - 1) ->
            ctx.MkITE(
                ctx.MkAnd(
                    ctx.MkGt(m.Eval(courseVariables.[i],true) :?> ArithExpr, ctx.
                        MkInt 0),
                    ctx.MkLe(m.Eval(courseVariables.[i],true) :?> ArithExpr, ctx.
                        MkInt maxScheduleCode)),
                ctx.MkEq(m.Eval(courseVariables.[i],true) :?> ArithExpr,
                    courseVariables.[i]),
                ctx.MkEq(courseVariables.[i], ctx.MkInt 0)) :?> BoolExpr
        |]
    ctx.MkAnd(boolArray)
```

Appendiks B

Model

Indhold

B.1 Datatyper

```
type CourseNumber = string

type CourseName = string

type ECTS = float

///Basic: 1, Advanced: 2, Phd: 3
type CourseLevel = int

type ScheduleCode = | E1 | E2 | E3 | E4 | E5 | F1 | F2 | F3 | F4 | F5
                    | E1A | E3A | E5A | E2B | E4B | E2A | E4A | E5B | E1B | E3B
                    | F1A | F3A | F5A | F2B | F4B | F2A | F4A | F5B | F1B | F3B
                    | January | June | E | F | Unknown

type PointBlock = Set<CourseNumber>
```

```
type CoursePrerequisitesDisjunctions = Set<CourseNumber>

type CoursePrerequisites = Set<CoursePrerequisitesDisjunctions>

type MandatoryCoursePrerequisites = CoursePrerequisites

type Field = string

type Fields = Set<Field>

type Course = CourseNumber * CourseName * ECTS * CourseLevel * ScheduleCode *
    PointBlock * MandatoryCoursePrerequisites * Fields

type InputCourse = CourseNumber * CourseName * ECTS * CourseLevel * Set<
    ScheduleCode> * PointBlock * MandatoryCoursePrerequisites * Fields

///Set<index * ScheduleCode>
type PartialCoursePairs = Set<int * ScheduleCode>


type StudyCourses = Set<CourseNumber>

type MandatoryStudyCourses = Set<CourseNumber>

type BasicScienceCourses = Set<CourseNumber>

type TechnologicalStudyCourses = Set<CourseNumber>

type ProjectsAndCommonCourses = Set<CourseNumber>

type StudyName = string

///Bachelor: 1, Master: 2
type StudyLevel = int

type BachelorStudy = StudyCourses * MandatoryStudyCourses * BasicScienceCourses *
    TechnologicalStudyCourses * ProjectsAndCommonCourses

type BachelorStudyDatabase = Map<StudyName,BachelorStudy>

type GeneralCompetenceCourses = Set<CourseNumber>

type TechnologicalSpecializationCourses = Set<CourseNumber>

type MasterStudy = StudyCourses * GeneralCompetenceCourses *
    TechnologicalSpecializationCourses

type MasterStudyDatabase = Map<StudyName,MasterStudy>


type CourseDatabase = Course array

type InputCourseDatabase = InputCourse array
```

```
type Semester = Set<Course * int * ScheduleCode * int * int>

type StudyPlan = Semester array


type CourseWantedSoftConstraint = CourseNumber * int

type CourseWantedSoftConstraints = Set<CourseWantedSoftConstraint>

type CourseWantedHardConstraint = CourseNumber * int

type CourseWantedHardConstraints = Set<CourseWantedHardConstraint>

type CourseUnwantedHardConstraint = CourseNumber * int

type CourseUnwantedHardConstraints = Set<CourseUnwantedHardConstraint>

type CourseConstraints = CourseWantedSoftConstraints * CourseWantedHardConstraints *
      CourseUnwantedHardConstraints


type ScheduleCodeConstraint = ScheduleCode * int

type ScheduleCodeUnwantedSoftConstraints = Set<ScheduleCodeConstraint>

type ScheduleCodeUnwantedHardConstraints = Set<ScheduleCodeConstraint>

type ScheduleCodeConstraints = ScheduleCodeUnwantedHardConstraints *
     ScheduleCodeUnwantedSoftConstraints


type StudyType = | Valid | Partial

type StudyLength = int

type ECTSDistribution = (ECTS * ECTS) array

type ThreeWeekPeriodConstraints = bool array

type StudyAbroadSemester = ECTS * int

type StudyAbroadSemesters = Set<StudyAbroadSemester>

type FieldRequirement = Field * ECTS

type FieldRequirements = FieldRequirement array


type SemesterStartInSpring = bool

type PassedCourses = Set<CourseNumber>

type CourseToIndex = Map<string, int>

type CourseNumberMap = Map<CourseNumber, Set<CourseNumber>>
```

```
let numberOfScheduleCodes = 34

let codeToInt c =
    match c with
    | E1 -> 1 | E1A -> 2 | E1B -> 3 | E2 -> 4 | E2A -> 5 | E2B -> 6
    | E3 -> 7 | E3A -> 8 | E3B -> 9 | E4 -> 10 | E4A -> 11 | E4B -> 12
    | E5 -> 13 | E5A -> 14 | E5B -> 15 | January -> 16 | E -> 17
    | F1 -> 18 | F1A -> 19 | F1B -> 20 | F2 -> 21 | F2A -> 22 | F2B -> 23
    | F3 -> 24 | F3A -> 25 | F3B -> 26 | F4 -> 27 | F4A -> 28 | F4B -> 29
    | F5 -> 30 | F5A -> 31 | F5B -> 32 | June -> 33 | F -> 34
    | Unknown -> 0

let intToCode i =
    match i with
    | 1 -> E1 | 2 -> E1A | 3 -> E1B | 4 -> E2 | 5 -> E2A | 6 -> E2B
    | 7 -> E3 | 8 -> E3A | 9 -> E3B | 10 -> E4 | 11 -> E4A | 12 -> E4B
    | 13 -> E5 | 14 -> E5A | 15 -> E5B | 16 -> January | 17 -> E
    | 18 -> F1 | 19 -> F1A | 20 -> F1B | 21 -> F2 | 22 -> F2A | 23 -> F2B
    | 24 -> F3 | 25 -> F3A | 26 -> F3B | 27 -> F4 | 28 -> F4A | 29 -> F4B
    | 30 -> F5 | 31 -> F5A | 32 -> F5B | 33 -> June | 34 -> F
    | _ -> Unknown

let stringToCode s =
    match s with
    | "E1" -> E1 | "E2" -> E2 | "E3" -> E3 | "E4" -> E4 | "E5" -> E5
    | "F1" -> F1 | "F2" -> F2 | "F3" -> F3 | "F4" -> F4 | "F5" -> F5
    | "E1A" -> E1A | "E2A" -> E2A | "E3A" -> E3A | "E4A" -> E4A | "E5A" ->
        E5A
    | "E1B" -> E1B | "E2B" -> E2B | "E3B" -> E3B | "E4B" -> E4B | "E5B" ->
        E5B
    | "F1A" -> F1A | "F2A" -> F2A | "F3A" -> F3A | "F4A" -> F4A | "F5A" ->
        F5A
    | "F1B" -> F1B | "F2B" -> F2B | "F3B" -> F3B | "F4B" -> F4B | "F5B" ->
        F5B
    | "January" -> January | "June" -> June | "E" -> E | "F" -> F
    | _ -> Unknown

let codeToString s =
    match s with
    | E1 -> "E1" | E2 -> "E2" | E3 -> "E3" | E4 -> "E4" | E5 -> "E5"
    | F1 -> "F1" | F2 -> "F2" | F3 -> "F3" | F4 -> "F4" | F5 -> "F5"
    | E1A -> "E1A" | E2A -> "E2A" | E3A -> "E3A" | E4A -> "E4A" | E5A -> "
        E5A"
    | E1B -> "E1B" | E2B -> "E2B" | E3B -> "E3B" | E4B -> "E4B" | E5B -> "
        E5B"
    | F1A -> "F1A" | F2A -> "F2A" | F3A -> "F3A" | F4A -> "F4A" | F5A -> "
        F5A"
    | F1B -> "F1B" | F2B -> "F2B" | F3B -> "F3B" | F4B -> "F4B" | F5B -> "
        F5B"
    | January -> "January" | June -> "June" | E -> "E" | F -> "F"
    | _ -> "Unknown"
```

B.2 Konstruktions-funktioner

```
/// Use a global context for creating constraints
let ctx = new Context()

/// Creates variables for all courses
let makeCourseVariables courseDatabaseLength =
    [| for i in 0..(courseDatabaseLength - 1) ->
           ctx.MkIntConst(sprintf "C_%i" i)
    |]

/// Creates ECTS points for all courses
let makeECTSPoints (courseDatabase:CourseDatabase) =
    [| for i in 0..((Array.length courseDatabase) - 1) ->
           let (_,_,p,_,_,_,_,_) = courseDatabase.[i]
           ctx.MkInt (int (p * 10.0))
    |]

/// Generates extended database
let extendDB (inputCourseDatabase:InputCourseDatabase) =
    /// Split a course
    let rec splitCourse (index : int) (numberOfScheduleCodes:int) (pcps:
        PartialCoursePairs) (courseDBList:List<Course>) (inputCourse:InputCourse) =
        let (courseNumber, courseName, ects, courseLevel, scs, pointBlock, mcps, fields)
            = inputCourse
        let scheduleCode = Set.minElement scs
        match Set.count scs with
        // Every split course gets an equal amount of ECTS points
        | 1 -> ((courseNumber, courseName, (ects / (float) numberOfScheduleCodes),
            courseLevel, scheduleCode, pointBlock, mcps, fields) :: courseDBList,
                Set.add (index, scheduleCode) pcps, index + 1)
        // The point block, prerequisites and fields values will only be checked for one
            of the split courses,
        // therefore only one of the courses gets these values
        | _ -> let newDBList = (courseNumber, courseName, (ects / (float)
            numberOfScheduleCodes), courseLevel, scheduleCode, Set.empty, Set.empty,
            Set.empty) :: courseDBList
               // The relation between split courses
               let newPcps = Set.add (index, scheduleCode) pcps
               splitCourse (index + 1) numberOfScheduleCodes newPcps newDBList
                           (courseNumber, courseName, ects, courseLevel, Set.remove
                               scheduleCode scs, pointBlock, mcps, fields)

    let rec makeLists (i : int) (index : int) (courseDbList : List<Course>) (pcpss : Set
        <PartialCoursePairs>) =
        if i = Array.length inputCourseDatabase
        then (courseDbList, pcpss)
        else let inputCourse = inputCourseDatabase.[i]
             let (_, _, _, _, scs, _, _, _) = inputCourse
             let numberOfScheduleCodes = Set.count scs
             if numberOfScheduleCodes = 1
             then let (newDbList, _, newIndex) = splitCourse index
                      numberOfScheduleCodes Set.empty courseDbList inputCourse
                  makeLists (i + 1) newIndex newDbList pcpss
```

```
            else let (newDbList, newPcps, newIndex) = splitCourse index
                 numberOfScheduleCodes Set.empty courseDbList inputCourse
                makeLists (i + 1) newIndex newDbList (Set.add newPcps pcpss)

    let (courseDbList, pcpss) = makeLists 0 0 [] Set.empty
    let courseDatabase = List.toArray (List.rev courseDbList)
    (courseDatabase, pcpss)

/// Creates a map from the course number to the index in the course database (and
    courseVariables)
let createCourseNumberToIndex (courseDatabase:CourseDatabase) =
    Map.ofList (List.rev[for i in 0..((Array.length courseDatabase) - 1) ->
                         let (courseNumber,_,_,_,_,_,_) = courseDatabase.[i]
                         (courseNumber, i)
                        ])
```

B.3 Løsnings-funktioner

```
/// Solution - Valid bachelor study plan
let rec solveValidBachelor (inputCourseDatabase:InputCourseDatabase) (courseConstraints
    :CourseConstraints)
                           (scheduleCodeConstraints:ScheduleCodeConstraints) (
                                ectsDistribution:ECTSDistribution)
                           (threeWeekPeriodHardConstraints:ThreeWeekPeriodConstraints
                                ) (studyLength:StudyLength)
                           (fieldRequirements:FieldRequirements) (studyName:StudyName
                                ) (passedCourses:PassedCourses)
                           (semesterStartInSpring:SemesterStartInSpring) (
                                courseNumberMap : CourseNumberMap)
                           (studyAbroadSemesters:StudyAbroadSemesters) =
    let solver = ctx.MkSolver()

    // (CourseDatabase, Set of PartialCoursePairs)
    let (courseDatabase, pcpss) = extendDB inputCourseDatabase
    let courseDatabaseLength = Array.length courseDatabase
    let courseVariables = makeCourseVariables (Array.length courseDatabase)
    let ectsPoints = makeECTSPoints courseDatabase
    let courseToIndex = createCourseNumberToIndex courseDatabase
    let bachelorStudy = Map.find studyName bachelorStudyDatabase
    let (studyCourses,_,_,_,_) = bachelorStudy
    let passedCoursesAlt =
        Set.fold (fun acc cn -> let set = Map.tryFind cn courseNumberMap
                                match set with
                                | Some cns -> Set.union cns acc
                                | _ -> acc) Set.empty passedCourses

    // (CourseWantedSoftConstraints, CourseWantedHardConstraints,
        CourseUnwantedHardConstraints)
    let (cwscs, cwhcs, cuhcs) = courseConstraints

    // (ScheduleCodeUnwantedHardConstraints, ScheduleCodeUnwantedSoftConstraints)
    let (scuhcs, scuscs) = scheduleCodeConstraints
```

```
/// Course soft constraints (VK8)
let rec courseSoftConstraint (cwscs:CourseWantedSoftConstraints) (solutionSet:
    CourseWantedHardConstraints) =
    match Set.count cwscs with
    | 0 -> solutionSet
    | _ -> let e = Set.maxElement cwscs
           let solutionSet' =
               let solution =
                   solveValidBachelor inputCourseDatabase (Set.empty, (Set.union
                       cwhcs (set[e])), cuhcs) (scuhcs, Set.empty)
                                      ectsDistribution
                                          threeWeekPeriodHardConstraints
                                          studyLength fieldRequirements
                                      studyName passedCourses
                                          semesterStartInSpring courseNumberMap
                                          studyAbroadSemesters
               match solution with
                   | None -> solutionSet
                   | Some _ -> courseSoftConstraint (Set.remove e cwscs) (Set.
                       union solutionSet (set[e]))

           if (Set.difference cwscs solutionSet') = Set.empty
           then
               if (Set.count solutionSet) >= (Set.count solutionSet')
               then solutionSet
               else solutionSet'
           else
               let recursiveSolutionSet = courseSoftConstraint (Set.remove e cwscs)
                   solutionSet
               if (Set.count recursiveSolutionSet) >= (Set.count solutionSet')
               then recursiveSolutionSet
               else solutionSet'

let cwhcsNew = Set.union cwhcs (courseSoftConstraint cwscs Set.empty)


/// Schedule code soft constraints (VK6)
let rec scheduleCodeSoftConstraint (scuscs:ScheduleCodeUnwantedSoftConstraints) (
    solutionSet:ScheduleCodeUnwantedHardConstraints) =
    match Set.count scuscs with
    | 0 -> solutionSet
    | _ -> let e = Set.maxElement scuscs
           let solutionSet' =
               let solution =
                   solveValidBachelor inputCourseDatabase (Set.empty, cwhcsNew,
                       cuhcs) (Set.union scuhcs (set[e]), Set.empty)
                                      ectsDistribution
                                          threeWeekPeriodHardConstraints
                                          studyLength fieldRequirements
                                      studyName passedCourses
                                          semesterStartInSpring courseNumberMap
                                          studyAbroadSemesters
               match solution with
                   | None -> solutionSet
```

```
                    | Some _ -> scheduleCodeSoftConstraint (Set.remove e scuscs) (
                        Set.union solutionSet (set[e]))

                if (Set.difference scuscs solutionSet') = Set.empty
                then
                    if (Set.count solutionSet) >= (Set.count solutionSet')
                    then solutionSet
                    else solutionSet'
                else
                    let recursiveSolutionSet = scheduleCodeSoftConstraint (Set.remove e
                        scuscs) solutionSet
                    if (Set.count recursiveSolutionSet) >= (Set.count solutionSet')
                    then recursiveSolutionSet
                    else solutionSet'

let scuhcsNew = Set.union scuhcs (scheduleCodeSoftConstraint scuscs Set.empty)


// General study plan constraints
solver.Assert(courseOverlapConstraint courseVariables courseDatabase)
solver.Assert(pointBlock courseVariables courseDatabase courseToIndex
    passedCoursesAlt courseNumberMap)
solver.Assert(coursePrerequisites courseVariables courseDatabase courseToIndex
    passedCoursesAlt courseNumberMap)
solver.Assert(rightAmountOfSemesters courseVariables studyLength
    semesterStartInSpring)
solver.Assert(studyAbroadConstraint courseVariables studyAbroadSemesters)
solver.Assert(passedCoursesConstraint courseVariables passedCoursesAlt
    courseDatabase courseToIndex)
solver.Assert(threeWeekPeriodConstraint courseVariables
    threeWeekPeriodHardConstraints)
solver.Assert(semesterPointConstraints courseVariables courseDatabaseLength
    ectsPoints ectsDistribution studyAbroadSemesters)
solver.Assert(freeScheduleCodes courseVariables scuhcs)
solver.Assert(selectedCourses courseVariables cwhcs courseToIndex courseNumberMap
    )
solver.Assert(deselectedCourses courseVariables cuhcs courseToIndex
    courseNumberMap)
solver.Assert(fieldRequirementsConstraint courseVariables courseDatabase
    fieldRequirements)

// Valid bachelor study plan
solver.Assert(bachelorPointConstraint courseVariables ectsPoints passedCoursesAlt
    courseDatabase studyAbroadSemesters)
solver.Assert(bachelorLengthConstraint studyLength)
solver.Assert(mandatoryBachelorCourses courseVariables bachelorStudy courseToIndex
    courseNumberMap passedCoursesAlt)
solver.Assert(bachelorCourseLevelConstraint courseVariables courseDatabase)
solver.Assert(courseTypeConstraint courseVariables courseDatabase bachelorStudy
    courseNumberMap passedCourses)
solver.Assert(bachelorProjectConstraint courseVariables courseToIndex ectsPoints
    passedCoursesAlt courseDatabase)

// General constraints
solver.Assert(partialCourseConstraint courseVariables pcpss courseDatabase)
```

```
    solver.Assert(sameScheduleCodeConstraint courseVariables courseDatabase)

    printfn "%s" "Solving bachelor..."
    let result = solver.Check()

    if result = Status.SATISFIABLE then
        let m = solver.Model
        let choices =
            set
                [for i in 0..(Array.length(courseVariables)-1) ->
                    let code = Convert.ToInt32(m.Evaluate(courseVariables.[i],true).
                        ToString())
                    let theCourse = courseDatabase.[i]
                    let semester = ((code - 1) / (numberOfScheduleCodes / 2)) + 1
                    let year = (semester + 1) / 2
                    let scheduleCode =
                        if year = 1
                        then intToCode code
                        else intToCode (code % (numberOfScheduleCodes * (year - 1)))
                    if code > 0
                    then (theCourse,semester,scheduleCode,code,i)
                    else (("","",0.0,1,Unknown,Set.empty,Set.empty,Set.empty),0,
                        Unknown,0,0)
                ]

        let studyPlan:StudyPlan =
            [| for semester in 1..studyLength ->
                Set.filter (fun (_,s,_,_,_) -> semester = s) choices
            |]

        Some(studyPlan,m)
    else
        None


/// Solution - Valid master study plan
let rec solveValidMaster (inputCourseDatabase:InputCourseDatabase) (courseConstraints:
    CourseConstraints)
                        (scheduleCodeConstraints:ScheduleCodeConstraints) (
                            ectsDistribution:ECTSDistribution)
                        (threeWeekPeriodHardConstraints:ThreeWeekPeriodConstraints) (
                            studyLength:StudyLength)
                        (fieldRequirements:FieldRequirements) (studyName:StudyName) (
                            passedCourses:PassedCourses)
                        (passedMasterCourses:PassedCourses) (semesterStartInSpring:
                            SemesterStartInSpring)
                        (courseNumberMap : CourseNumberMap) (studyAbroadSemesters
                            :StudyAbroadSemesters) =

    let solver = ctx.MkSolver()

    // (CourseDatabase, Set of PartialCoursePairs)
    let (courseDatabase, pcpss) = extendDB inputCourseDatabase
    let courseDatabaseLength = Array.length courseDatabase
    let courseVariables = makeCourseVariables (Array.length courseDatabase)
```

```
let ectsPoints = makeECTSPoints courseDatabase
let courseToIndex = createCourseNumberToIndex courseDatabase
let masterStudy = Map.find studyName masterStudyDatabase
let (studyCourses,_,_) = masterStudy
let passedCoursesAlt =
    Set.fold (fun acc cn -> let set = Map.tryFind cn courseNumberMap
                            match set with
                            | Some cns -> Set.union cns acc
                            | _ -> acc) Set.empty passedCourses
let passedMasterCoursesAlt =
    Set.fold (fun acc cn -> let set = Map.tryFind cn courseNumberMap
                            match set with
                            | Some cns -> Set.union cns acc
                            | _ -> acc) Set.empty passedMasterCourses

// (CourseUnwantedHardConstraints, CourseWantedHardConstraints,
     CourseWantedSoftConstraints)
let (cwscs, cwhcs, cuhcs) = courseConstraints

// (ScheduleCodeUnwantedHardConstraints, ScheduleCodeUnwantedSoftConstraints)
let (scuhcs, scuscs) = scheduleCodeConstraints

/// Course soft constraints (VK8)
let rec courseSoftConstraint (cwscs:CourseWantedSoftConstraints) (solutionSet:
     CourseWantedHardConstraints) =
    match Set.count cwscs with
    | 0 -> solutionSet
    | _ -> let e = Set.maxElement cwscs
           let solutionSet' =
               let solution =
                   solveValidMaster inputCourseDatabase (Set.empty, (Set.union
                         cwhcs (set[e])), cuhcs) (scuhcs, Set.empty)
                                    ectsDistribution
                                        threeWeekPeriodHardConstraints
                                        studyLength fieldRequirements
                                    studyName passedCourses passedMasterCourses
                                        semesterStartInSpring courseNumberMap
                                    studyAbroadSemesters
               match solution with
                   | None -> solutionSet
                   | Some _ -> courseSoftConstraint (Set.remove e cwscs) (Set.
                       union solutionSet (set[e]))

           if (Set.difference cwscs solutionSet') = Set.empty
           then
               if (Set.count solutionSet) >= (Set.count solutionSet')
               then solutionSet
               else solutionSet'
           else
               let recursiveSolutionSet = courseSoftConstraint (Set.remove e cwscs)
                    solutionSet
               if (Set.count recursiveSolutionSet) >= (Set.count solutionSet')
               then recursiveSolutionSet
               else solutionSet'
```

```
let cwhcsNew = Set.union cwhcs (courseSoftConstraint cwscs Set.empty)


/// Schedule code soft constraints (VK6)
let rec scheduleCodeSoftConstraint (scuscs:ScheduleCodeUnwantedSoftConstraints) (
    solutionSet:ScheduleCodeUnwantedHardConstraints) =
    match Set.count scuscs with
    | 0 -> solutionSet
    | _ -> let e = Set.maxElement scuscs
           let solutionSet' =
               let solution =
                   solveValidMaster inputCourseDatabase (Set.empty, cwhcsNew,
                       cuhcs) (Set.union scuhcs (set[e]), Set.empty)
                                   ectsDistribution
                                       threeWeekPeriodHardConstraints
                                       studyLength fieldRequirements
                                   studyName passedCourses passedMasterCourses
                                       semesterStartInSpring courseNumberMap
                                   studyAbroadSemesters
               match solution with
                   | None -> solutionSet
                   | Some _ -> scheduleCodeSoftConstraint (Set.remove e scuscs) (
                       Set.union solutionSet (set[e]))

           if (Set.difference scuscs solutionSet') = Set.empty
           then
               if (Set.count solutionSet) >= (Set.count solutionSet')
               then solutionSet
               else solutionSet'
           else
               let recursiveSolutionSet = scheduleCodeSoftConstraint (Set.remove e
                   scuscs) solutionSet
               if (Set.count recursiveSolutionSet) >= (Set.count solutionSet')
               then recursiveSolutionSet
               else solutionSet'

let scuhcsNew = Set.union scuhcs (scheduleCodeSoftConstraint scuscs Set.empty)


// General study plan constraints
solver.Assert(courseOverlapConstraint courseVariables courseDatabase)
solver.Assert(pointBlock courseVariables courseDatabase courseToIndex (Set.union
    passedCoursesAlt passedMasterCoursesAlt) courseNumberMap)
solver.Assert(coursePrerequisites courseVariables courseDatabase courseToIndex (Set.
    union passedCoursesAlt passedMasterCoursesAlt) courseNumberMap)
solver.Assert(rightAmountOfSemesters courseVariables studyLength
    semesterStartInSpring)
solver.Assert(studyAbroadConstraint courseVariables studyAbroadSemesters)
solver.Assert(passedCoursesConstraint courseVariables (Set.union passedCoursesAlt
    passedMasterCoursesAlt) courseDatabase courseToIndex)
solver.Assert(threeWeekPeriodConstraint courseVariables
    threeWeekPeriodHardConstraints)
solver.Assert(semesterPointConstraints courseVariables courseDatabaseLength
    ectsPoints ectsDistribution studyAbroadSemesters)
solver.Assert(freeScheduleCodes courseVariables scuhcsNew)
```

```
    solver.Assert(selectedCourses courseVariables cwhcsNew courseToIndex
        courseNumberMap)
    solver.Assert(deselectedCourses courseVariables cuhcs courseToIndex
        courseNumberMap)
    solver.Assert(fieldRequirementsConstraint courseVariables courseDatabase
        fieldRequirements)

    // Valid master study plan
    solver.Assert(masterPointConstraint courseVariables ectsPoints
        passedMasterCoursesAlt courseDatabase studyAbroadSemesters)
    solver.Assert(masterLengthConstraint studyLength)
    solver.Assert(masterCourseLevelConstraint courseVariables courseDatabase)
    solver.Assert(masterStudyCourseDistribution courseVariables courseDatabase
        studyName masterStudy courseToIndex courseNumberMap
                                                (Set.union passedCoursesAlt
                                                    passedMasterCoursesAlt))
    solver.Assert(masterThesisConstraint courseVariables courseToIndex ectsPoints
        passedMasterCoursesAlt courseDatabase)

    // General constraints
    solver.Assert(partialCourseConstraint courseVariables pcpss courseDatabase)
    solver.Assert(sameScheduleCodeConstraint courseVariables courseDatabase)

    printfn "%s" "Solving master..."
    let result = solver.Check()

    if result = Status.SATISFIABLE then
        let m = solver.Model
        let choices =
            set
                [for i in 0..(Array.length(courseVariables)-1) ->
                    let code = Convert.ToInt32(m.Evaluate(courseVariables.[i],true).
                        ToString())
                    let theCourse = courseDatabase.[i]
                    let semester = ((code - 1) / (numberOfScheduleCodes / 2)) + 1
                    let year = (semester + 1) / 2
                    let scheduleCode =
                        if year = 1
                        then intToCode code
                        else intToCode (code % (numberOfScheduleCodes * (year - 1)))
                    if code > 0
                    then (theCourse,semester,scheduleCode,code,i)
                    else (("","",0.0,1, Unknown,Set.empty,Set.empty),0,
                        Unknown,0,0)
                ]

        let studyPlan:StudyPlan =
            [|for semester in 1..studyLength ->
                Set.filter (fun (_,s,_,_,_) -> semester = s) choices
            |]
        Some(studyPlan,m)
    else
        None

/// Solution - Partial study plan
```

```
let rec solvePartial (inputCourseDatabase:InputCourseDatabase) (courseConstraints:
    CourseConstraints)
                    (scheduleCodeConstraints:ScheduleCodeConstraints) (ectsDistribution:
                        ECTSDistribution)
                    (threeWeekPeriodHardConstraints:ThreeWeekPeriodConstraints) (
                        studyLength:StudyLength)
                    (fieldRequirements:FieldRequirements) (studyName:StudyName) (
                        passedCourses:PassedCourses)
                    (passedMasterCourses:PassedCourses) (semesterStartInSpring:
                        SemesterStartInSpring)
                    (courseNumberMap : CourseNumberMap) (studyAbroadSemesters:
                        StudyAbroadSemesters) =
    let solver = ctx.MkSolver()

    // (CourseDatabase, Set of PartialCoursePairs)
    let (courseDatabase, pcpss) = extendDB inputCourseDatabase
    let courseDatabaseLength = Array.length courseDatabase
    let courseVariables = makeCourseVariables (Array.length courseDatabase)
    let ectsPoints = makeECTSPoints courseDatabase
    let courseToIndex = createCourseNumberToIndex courseDatabase
    let studyLevel = Map.find studyName studyLevelDatabase
    let passedCoursesAlt =
        Set.fold (fun acc cn -> let set = Map.tryFind cn courseNumberMap
                                match set with
                                | Some cns -> Set.union cns acc
                                | _ -> acc) Set.empty passedCourses
    let passedMasterCoursesAlt =
        Set.fold (fun acc cn -> let set = Map.tryFind cn courseNumberMap
                                match set with
                                | Some cns -> Set.union cns acc
                                | _ -> acc) Set.empty passedMasterCourses

    // (CourseUnwantedHardConstraints, CourseWantedHardConstraints,
        CourseWantedSoftConstraints)
    let (cwscs, cwhcs, cuhcs) = courseConstraints

    // (ScheduleCodeUnwantedHardConstraints, ScheduleCodeUnwantedSoftConstraints)
    let (scuhcs, scuscs) = scheduleCodeConstraints

    /// Course soft constraints (VK8)
    let rec courseSoftConstraint (cwscs:CourseWantedSoftConstraints) (solutionSet:
        CourseWantedHardConstraints) =
        match Set.count cwscs with
        | 0 -> solutionSet
        | _ -> let e = Set.maxElement cwscs
               let solutionSet' =
                   let solution =
                       solvePartial inputCourseDatabase (Set.empty, (Set.union cwhcs (
                           set[e])), cuhcs) (scuhcs, Set.empty)
                                    ectsDistribution threeWeekPeriodHardConstraints
                                        studyLength fieldRequirements
                                    studyName passedCourses passedMasterCourses
                                        semesterStartInSpring courseNumberMap
                                    studyAbroadSemesters
                   match solution with
```

```
                    | None -> solutionSet
                    | Some _ -> courseSoftConstraint (Set.remove e cwscs) (Set.
                        union solutionSet (set[e]))

            if (Set.difference cwscs solutionSet') = Set.empty
            then
                if (Set.count solutionSet) >= (Set.count solutionSet')
                then solutionSet
                else solutionSet'
            else
                let recursiveSolutionSet = courseSoftConstraint (Set.remove e cwscs)
                    solutionSet
                if (Set.count recursiveSolutionSet) >= (Set.count solutionSet')
                then recursiveSolutionSet
                else solutionSet'

let cwhcsNew = Set.union cwhcs (courseSoftConstraint cwscs Set.empty)


/// Schedule code soft constraints (VK6)
let rec scheduleCodeSoftConstraint (scuscs:ScheduleCodeUnwantedSoftConstraints) (
    solutionSet:ScheduleCodeUnwantedHardConstraints) =
    match Set.count scuscs with
    | 0 -> solutionSet
    | _ -> let e = Set.maxElement scuscs
           let solutionSet' =
               let solution =
                   solvePartial inputCourseDatabase (Set.empty, cwhcsNew, cuhcs) (
                       Set.union scuhcs (set[e]), Set.empty)
                               ectsDistribution threeWeekPeriodHardConstraints
                                   studyLength fieldRequirements
                               studyName passedCourses passedMasterCourses
                                   semesterStartInSpring courseNumberMap
                               studyAbroadSemesters
               match solution with
                   | None -> solutionSet
                   | Some _ -> scheduleCodeSoftConstraint (Set.remove e scuscs) (
                       Set.union solutionSet (set[e]))

           if (Set.difference scuscs solutionSet') = Set.empty
           then
               if (Set.count solutionSet) >= (Set.count solutionSet')
               then solutionSet
               else solutionSet'
           else
               let recursiveSolutionSet = scheduleCodeSoftConstraint (Set.remove e
                   scuscs) solutionSet
               if (Set.count recursiveSolutionSet) >= (Set.count solutionSet')
               then recursiveSolutionSet
               else solutionSet'

let scuhcsNew = Set.union scuhcs (scheduleCodeSoftConstraint scuscs Set.empty)

// General study plan constraints
```

```
solver.Assert(selectedCourses courseVariables cwhcsNew courseToIndex
    courseNumberMap)
solver.Assert(fieldRequirementsConstraint courseVariables courseDatabase
    fieldRequirements)

// General constraints
solver.Assert(partialCourseConstraint courseVariables pcpss courseDatabase)

// Partial study plan constraints
solver.Assert(expandableToValid courseVariables studyLevel inputCourseDatabase
    courseConstraints scheduleCodeConstraints
                                ectsDistribution threeWeekPeriodHardConstraints
                                    studyLength fieldRequirements
                                studyName passedCourses passedMasterCourses
                                    semesterStartInSpring
                                courseNumberMap studyAbroadSemesters)

printfn "%s" "Solving partial..."
let result = solver.Check()

if result = Status.SATISFIABLE then
    let m = solver.Model
    let choices =
        set
            [ for i in 0..(Array.length(courseVariables)-1) ->
                let code = Convert.ToInt32(m.Evaluate(courseVariables.[i],true).
                    ToString())
                let theCourse = courseDatabase.[i]
                let semester = ((code - 1) / (numberOfScheduleCodes / 2)) + 1
                let year = (semester + 1) / 2
                let scheduleCode =
                    if year = 1
                    then intToCode code
                    else intToCode (code % (numberOfScheduleCodes * (year - 1)))
                if code > 0
                then (theCourse,semester,scheduleCode,code,i)
                else (("","",0.0,1,Unknown,Set.empty,Set.empty,Set.empty),0,
                    Unknown,0,0)
            ]

    let studyPlan:StudyPlan =
        [| for semester in 1..studyLength ->
            Set.filter (fun (_,s,_,_,_) -> semester = s) choices
        |]

    Some(studyPlan,m)
else
    None
```

Appendiks C

Databaser

Indhold

C.1 Kursusdatabasen

C.1.1 XMLParser

```
let  filter  = set ["01005"]

let readXmlFile (xmlFile:string) (courseSet:Set<CourseNumber>) =
    use stringReader = new StringReader(xmlFile)
    use reader = XmlReader.Create stringReader

    let readCourseNumber (reader : XmlReader) =
        reader.GetAttribute("CourseCode")
```

```
let rec readCourseName (reader : XmlReader) =
    reader.GetAttribute("Title")

let readCourseLevel (reader : XmlReader) =
    match reader.GetAttribute("TypeKey") with
    | "DTU_Basic" -> 1
    | "DTU_Advanced" -> 2
    | _ -> 3

let readECTS (reader : XmlReader) =
    reader.ReadElementContentAsDouble()

let rec readScheduleCode (reader : XmlReader) (conList : List<ScheduleCode>) =
    if (reader.ReadToFollowing("Schedule"))
    then let sc = stringToCode (reader.GetAttribute("ScheduleKey"))
         let newConList = sc :: conList
         readScheduleCode reader newConList
    else (conList, reader)

let readPointBlock (reader : XmlReader) =
    let s = reader.GetAttribute("CourseCode")
    if String.exists (fun c -> c = '.') s
    then let newS = String.collect (fun c -> if c = ' ' then "" else Char.ToString c)
         s
         let stringArray = newS.Split [|'.'|]
         set [for cn in stringArray -> cn]
    else if String.exists (fun c -> c = '/') s
         then let newS = String.collect (fun c -> if c = ' ' then "" else Char.
              ToString c) s
              let stringArray = newS.Split [|'/'|]
              set [for cn in stringArray -> cn ]
         else let stringArray = s.Split [|' '|]
              set [for cn in stringArray -> cn]

let readMandatoryPrereq (reader : XmlReader) =
    let disStringToSet (disString : string) : CoursePrerequisitesDisjunctions =
        let disArray = disString.Split [|'/'|]
        set (Array.toList (disArray))

    let s = reader.GetAttribute("Txt")
    let newS = String.collect (fun c -> if c = ' ' then "" else Char.ToString c) s
    let conArray = newS.Split [|'.'|]
    let conSet = set (Array.toList (conArray))
    Set.map (fun e -> disStringToSet e) conSet

let rec readCourse (reader : XmlReader) inputCourse =
    let (number, name, ects, level, scheduleCodeList, pointBlock, mandPrereq, fields)
        = inputCourse

    if reader.Read()
    then match reader.Name with
         | "Course" when reader.IsStartElement() ->
                let newNumber = readCourseNumber reader
```

```
                // We don't want the original courses of the ones we construct our
                    own versions of, to be read into our database
                // Likewise we don't want any irrelevant courses either
                if Set.contains newNumber filter || not (Set.contains newNumber
                    courseSet)
                then ignore (reader.ReadToFollowing("EndOfSubtree"))
                    inputCourse
                else readCourse reader (newNumber, name, ects, level,
                    scheduleCodeList, pointBlock, mandPrereq, fields)
            | "Title" when reader.GetAttribute("Lang") = "dk" ->
                let newName = readCourseName reader
                readCourse reader (number, newName, ects, level, scheduleCodeList,
                    pointBlock, mandPrereq, fields)
            | "Point" ->
                let newEcts = readECTS reader
                readCourse reader (number, name, newEcts, level, scheduleCodeList,
                    pointBlock, mandPrereq, fields)
            | "Type" when reader.IsStartElement() ->
                let newLevel = readCourseLevel reader
                readCourse reader (number, name, ects, newLevel, scheduleCodeList,
                    pointBlock, mandPrereq, fields)
            | "Class_Schedule" ->
                let (newScheduleCodeList, _) = readScheduleCode (reader.
                    ReadSubtree()) []
                let newScheduleCodeList' = newScheduleCodeList :: scheduleCodeList
                readCourse reader (number, name, ects, level, newScheduleCodeList',
                    pointBlock, mandPrereq, fields)
            | "No_Credit_Points_With" when reader.MoveToAttribute("CourseCode")
                ->
                let newPointBlock = readPointBlock reader
                readCourse reader (number, name, ects, level, scheduleCodeList,
                    newPointBlock, mandPrereq, fields)
            | "Mandatory_Prerequisites" when reader.ReadToDescendant("
                DTU_CoursesTxt") ->
                let newMandPrereq = readMandatoryPrereq reader
                readCourse reader (number, name, ects, level, scheduleCodeList,
                    pointBlock, newMandPrereq, fields)
            | _ -> readCourse reader inputCourse
        else inputCourse


    let rec read (reader : XmlReader) (courseList : List<InputCourse>) (
        courseNumberList : List<CourseNumber * Set<CourseNumber>>) =
        if reader.Read() then
            match reader.Name with
            | "Courses" ->
                let subtree = reader.ReadSubtree()
                let course = readCourse subtree ("", "", 0.0, 0, [], Set.empty, Set.
                    empty, Set.empty)
                let (number, name, ects, level, scheduleCodeList, pointBlock,
                    mandPrereq, fields) = course
                ignore (subtree.Close)

                // Some courses have disjunct schedule codes and are therefore split
                    up in different courses, one per schedule code
```

```
                let rec splitDisjunctScheduleCodes (index : int) (pointBlockSet : Set
                    <CourseNumber>) (scheduleCodeList : List<List<ScheduleCode
                    >>)
                                                    (courseList : List<InputCourse>)
                                                        (altCourseNumbers : List<
                                                        CourseNumber>) (
                                                        altCourseNumberList : List<
                                                        CourseNumber * Set<
                                                        CourseNumber>>) =
                    match scheduleCodeList with
                    | head :: tail ->
                            let scheduleCodeSet = set head
                            let newNumber = number + Convert.ToString(Convert.
                                ToChar(index))
                            splitDisjunctScheduleCodes (index + 1) (Set.add
                                newNumber pointBlockSet) tail
                                ((newNumber, name, ects, level, scheduleCodeSet,
                                    Set.union pointBlock pointBlockSet,
                                    mandPrereq, fields) :: courseList)
                                (newNumber :: altCourseNumbers) ((newNumber,
                                    set [newNumber]) :: altCourseNumberList)
                    | [] -> (courseList, set altCourseNumbers, altCourseNumberList)

                match scheduleCodeList with
                | [] ->          read reader courseList courseNumberList
                | head :: [] ->
                        let newCourse = (number, name, ects, level, set head,
                            pointBlock, mandPrereq, fields)
                        read reader (newCourse :: courseList) ((number, set [number])
                            :: courseNumberList)
                | head :: tail ->
                        let (newCourseList, altCourseNumbers, altCourseNumberList)
                            = splitDisjunctScheduleCodes 65 Set.empty
                            scheduleCodeList courseList [] []
                        read reader newCourseList ((number,altCourseNumbers) :: (
                            altCourseNumberList @ courseNumberList))

            | _ -> read reader courseList courseNumberList
        else (courseList, courseNumberList)

    let (courseList,courseNumberList) = read reader [] []
    let courseNumberMap = Map.ofList courseNumberList
    let courseNumberMapFinal = addToMap courseNumberMap
    let inputCourseDatabase = List.toArray (addSpecialCourses courseList)
    (inputCourseDatabase, courseNumberMapFinal)
```

C.1.2 Specielle kurser

```
let specialCourses : List<InputCourse> =
    [("01005A", "Matematik 1", 17.5, 1, set [E1A; E2; F1A; F2],
      set ["01000"; "01001"; "01002"; "01003"; "01007"; "01010"; "01011";
           "01012"; "01013"; "01014"; "01020"; "01021"; "01005B"; "01005C"],
      Set.empty, Set.empty);
     ("01005B", "Matematik 1", 17.5, 1, set [E3A; E4; F3A; F4],
```

```
      set ["01000"; "01001"; "01002"; "01003"; "01007"; "01010"; "01011";
           "01012"; "01013"; "01014"; "01020"; "01021"; "01005A"; "01005C"],
      Set.empty, Set.empty);
     ("01005C", "Matematik 1", 17.5, 1, set [E3B; E5; F3B; F5],
      set ["01000"; "01001"; "01002"; "01003"; "01007"; "01010"; "01011";
           "01012"; "01013"; "01014"; "01020"; "01021"; "01005A"; "01005B"],
      Set.empty, Set.empty);
     ("Bac15E", "Bachelor", 15.0, 1, set [E;January], set ["Bac20E";"Bac20F";"Bac15F"],
           Set.empty, Set.empty);
     ("Bac15F", "Bachelor", 15.0, 1, set [F;June], set ["Bac20E";"Bac20F";"Bac15E"],
          Set.empty, Set.empty);
     ("Bac20E", "Bachelor", 20.0, 1, set [E;January], set ["Bac20F";"Bac15F";"Bac15E"],
           Set.empty, Set.empty);
     ("Bac20F", "Bachelor", 20.0, 1, set [F;June], set ["Bac20E";"Bac15F";"Bac15E"],
          Set.empty, Set.empty);
     ("Mas30E", "Speciale", 30.0, 3, set [E;January], set ["Mas30F";"Mas32E";"Mas32F"
          ;"Mas35E";"Mas35F"], Set.empty, Set.empty);
     ("Mas30F", "Speciale", 30.0, 3, set [F;June], set ["Mas30E";"Mas32E";"Mas32F";"
          Mas35E";"Mas35F"], Set.empty, Set.empty);
     ("Mas32E", "Speciale", 32.5, 3, set [E;January], set ["Mas30E";"Mas30F";"Mas32F"
          ;"Mas35E";"Mas35F"], Set.empty, Set.empty);
     ("Mas32F", "Speciale", 32.5, 3, set [F;June], set ["Mas30E";"Mas30F";"Mas32E";"
          Mas35E";"Mas35F"], Set.empty, Set.empty);
     ("Mas35E", "Speciale", 35.0, 3, set [E;January], set ["Mas30E";"Mas30F";"Mas32E"
          ;"Mas32F";"Mas35F"], Set.empty, Set.empty);
     ("Mas35F", "Speciale", 35.0, 3, set [F;June], set ["Mas30E";"Mas30F";"Mas32E";"
          Mas32F";"Mas35E"], Set.empty, Set.empty)]

let specialCourseListForMapping =
      [("01005",set ["01005A";"01005B";"01005C"]);("Bac15",set["Bac15E";"Bac15F"]);(
           "Bac20",set["Bac20E";"Bac20F"]);
       ("Mas30",set["Mas30F";"Mas30E"]);("Mas32",set["Mas32F";"Mas32E"]);
       ("Mas35",set["Mas35F";"Mas35E"]);
       ("01005A",set["01005A"]);("01005B",set["01005B"]);("01005C",set["01005C"]);
       ("Bachelor",set["Bac15E";"Bac15F";"Bac20E";"Bac20F"]);("Speciale",set["
           Mas30E";"Mas30F";"Mas32E";"Mas32F";"Mas35E";"Mas35F"])]

let addSpecialCourses (courseList : List<InputCourse>) =
    let rec addSpecialCourses' (courseList : List<InputCourse>) (specialCourseList : List
        <InputCourse>) =
      match specialCourseList with
      | [] -> courseList
      | head :: tail -> addSpecialCourses' (head :: courseList) tail
    addSpecialCourses' courseList specialCourses

let addToMap (map:CourseNumberMap) =
    let rec addToMap' (map:CourseNumberMap) (list:List<CourseNumber * Set<
        CourseNumber>>) =
      match list with
      | [] -> map
      | head :: tail -> let (key,value) = head
                        addToMap' (Map.add key value map) tail
    addToMap' map specialCourseListForMapping
```

C.2 Studieretningsdatabase

C.2.1 Bachelor-database

```
let softwareTechnologyStudyName = "Softwareteknologi"

let softwareTechnologyStudyCourses =
    set ["02110";"01227";"01018";"02249";"02282";"02162";"02266";"02165";"02350";"
         02241";"02811";"02821";"01410";
         "02230";"02233";"02180";"02281";"02285";"02502";"02561";"02562";"02564";"
              02565";"02576";"30510";"30530";
         "30550";"42101";"42112";"42114";"42115";"02601";"02631";"02633";"02526";"
              42415";"42430";"02821";"34310";
         "34311";"34341";"30010";"31070";"31385";"30300";
         "01005";"01008";"10020";"26027";"01017";"01035";"02402";"02405";"10044";"
              26202";"27002";
         "02105";"02131";"02141";"02161";"02155";"02156";"02157";"02158";"02159";
         "02101";"02121";"02122";"42610"]

let softwareTechnologyMandatoryStudyCourses =
    set ["01005";"01008";"10020";"26027";
         "02105";"02131";"02141";"02161";
         "02101";"02121";"02122";"42610"]

let softwareTechnologyBasicScienceCourses =
    set ["01005";"01008";"10020";"26027";"01017";"01035";"02402";"02405";"10044";"
         26202";"27002"]

let softwareTechnologyTechnologicalStudyCourses =
    set ["02105";"02131";"02141";"02161";"02155";"02156";"02157";"02158";"02159"]

let softwareTechnologyProjectsAndCommonCourses =
    set ["02101";"02121";"02122";"42610"]

let softwareTechnologyStudy =
    (softwareTechnologyStudyCourses, softwareTechnologyMandatoryStudyCourses,
          softwareTechnologyBasicScienceCourses,
     softwareTechnologyTechnologicalStudyCourses,
           softwareTechnologyProjectsAndCommonCourses)


let bachelorStudyDatabase:BachelorStudyDatabase =
    Map.ofList [(softwareTechnologyStudyName, softwareTechnologyStudy)]
```

C.2.2 Kandidat-database

```
let computerScienceAndEngineeringStudyName = "Informationsteknologi"

let computerScienceAndEngineeringStudyCourses =
    set ["02110";"01227";"01018";"02249";"02282";"02162";"02266";"02165";"02350";"
         02241";"02811";"02821";"01410";
```

```
        "02230";"02233";"02180";"02281";"02285";"02502";"02561";"02562";"02564";"
            02565";"02576";"30510";"30530";
        "30550";"42101";"42112";"42114";"42115";"02601";"02631";"02633";"02526";"
            42415";"42430";"02821";"34310";
        "34311";"34341";"30010";"31070";"31385";"30300";
        "02105";"02131";"02141";"02161";"02155";"02156";"02157";"02158";"02159";
        "42435";"42490";"02203";"02205";"02220";"02223";"02239";"02242";"02249";"
            02285";"02291";
        "02203";"02204";"02205";"02209";"02211";"02213";"02217";"02220";"02223";"
            02224";"02228";"02232";"02233";
        "02234";"02238";"02239";"02241";"02244";"02246";"02249";"02257";"02263";"
            02264";"02265";"02267";"02281";
        "02282";"02284";"02285";"02286";"02291"]

let computerScienceAndEngineeringGeneralCompetenceCourses =
    set ["02203";"02205";"02220";"02223";"02239";"02242";"02249";"02285";"02291"]

let computerScienceAndEngineeringTechnologicalSpecializationCourses =
    set ["02203";"02204";"02205";"02209";"02211";"02213";"02217";"02220";"02223";"
        02224";"02228";"02232";"02233";
        "02234";"02238";"02239";"02241";"02242";"02244";"02246";"02249";"02257";"
            02263";"02264";"02265";"02267";
        "02281";"02282";"02284";"02285";"02286";"02291"]

let computerScienceAndEngineeringStudy =
    (computerScienceAndEngineeringStudyCourses,
          computerScienceAndEngineeringGeneralCompetenceCourses,
     computerScienceAndEngineeringTechnologicalSpecializationCourses)


let masterStudyDatabase:MasterStudyDatabase =
    Map.ofList [(computerScienceAndEngineeringStudyName,
          computerScienceAndEngineeringStudy)]


let studyLevelDatabase =
    Map.ofList [(softwareTechnologyStudyName, 1);
                (computerScienceAndEngineeringStudyName, 2)]
```

Appendiks D

Bruger-interaktion

Indhold

D.1 Læs fra fil

```
let readInput filename =
    let reader = new StreamReader( __SOURCE_DIRECTORY__ + "\\" + filename)
    let inputArray =
        [| for i in 0..13 -> reader.ReadLine()|]
    reader.Close()

    let studyType = match inputArray.[0] with
                    | "Gyldig" -> Valid
                    | _ -> Partial

    let studyName = inputArray.[1]

    let studyLength = Convert.ToInt32 inputArray.[2]

    let pointDistArray = (inputArray.[3]).Split    [|','|]
    let pointDist =
```

```
    [| for i in 0..(studyLength - 1) ->
        if (Array.length pointDistArray - 1) < i || pointDistArray.[i] = ""
        then (30.0,30.0)
        else
            let string = pointDistArray.[i]
            if String.exists (fun c -> c = '-') string
            then
                let minMaxArray = string.Split [|'-'|]
                let min = Convert.ToDouble minMaxArray.[0]
                let max = Convert.ToDouble minMaxArray.[1]
                (min,max)
            else
                let value = Convert.ToDouble string
                (value,value)
    |]

let startSeason = match inputArray.[4].Substring(0,3) with
                  | "For" -> true
                  | _ -> false

let threeWeekDistArray = (inputArray.[5]).Split [|','|]
let threeWeekDist =
    [| for i in 0..(studyLength - 1) ->
        if (Array.length threeWeekDistArray - 1) < i || threeWeekDistArray.[i] = ""
        then true
        else
            match threeWeekDistArray.[i] with
            | "nej" -> false
            | _ -> true
    |]

let cwhcsStringArray = (inputArray.[6]).Split [|','|]
let cwhcsArray =
    match cwhcsStringArray.[0] with
    | "" -> [||]
    | _ -> Array.map (fun (string : string) -> let stringArray = string.Split [|'-'|]
                                               let course = stringArray.[0]
                                               let semester = Convert.ToInt32
                                                   stringArray.[1]
                                               (course,semester)
                     ) cwhcsStringArray
let cwhcs = set (Array.toList cwhcsArray)

let cwscsStringArray = (inputArray.[7]).Split [|','|]
let cwscsArray =
    match cwscsStringArray.[0] with
    | "" -> [||]
    | _ -> Array.map (fun (string : string) -> let stringArray = string.Split [|'-'|]
                                               let course = stringArray.[0]
                                               let semester = Convert.ToInt32
                                                   stringArray.[1]
                                               (course,semester)
                     ) cwscsStringArray
let cwscs = set (Array.toList cwscsArray)
```

```
let cuhcsStringArray = (inputArray.[8]).Split    [|','|]
let cuhcsArray =
    match cuhcsStringArray.[0] with
    | "" -> [||]
    | _ -> Array.map (fun (string : string) -> let stringArray = string.Split   [|'-'|]
                                               let course = stringArray.[0]
                                               let semester = Convert.ToInt32
                                                   stringArray.[1]
                                               (course,semester)
                     ) cuhcsStringArray
let cuhcs = set (Array.toList cuhcsArray)

let scuhcStringArray = (inputArray.[9]).Split    [|','|]
let scuhcArray =
    match scuhcStringArray.[0] with
    | "" -> [||]
    | _ -> Array.map (fun (string : string) -> let stringArray = string.Split   [|'-'|]
                                               let scheduleCode = stringToCode
                                                   stringArray.[0]
                                               let semester = Convert.ToInt32
                                                   stringArray.[1]
                                               (scheduleCode,semester)
                     ) scuhcStringArray
let scuhcs = set (Array.toList scuhcArray)

let scuscStringArray = (inputArray.[10]).Split    [|','|]
let scuscArray =
    match scuscStringArray.[0] with
    | "" -> [||]
    | _ -> Array.map (fun (string : string) -> let stringArray = string.Split   [|'-'|]
                                               let scheduleCode = stringToCode
                                                   stringArray.[0]
                                               let semester = Convert.ToInt32
                                                   stringArray.[1]
                                               (scheduleCode,semester)
                     ) scuscStringArray
let scuscs = set (Array.toList scuscArray)

let semestersAbroadStringArray = (inputArray.[11]).Split   [|','|]
let semestersAbroadArray =
    match semestersAbroadStringArray.[0] with
    | "" -> [||]
    | _ -> Array.map (fun (string : string) -> let stringArray = string.Split   [|'-'|]
                                               let ects = Convert.ToDouble
                                                   stringArray.[0]
                                               let semester = Convert.ToInt32
                                                   stringArray.[1]
                                               (ects,semester)
                                    ) semestersAbroadStringArray
let semestersAbroad = set (Array.toList semestersAbroadArray)

let passedBachelorCoursesArray = inputArray.[12].Split   [|','|]
let passedBachelorCourses = set (Array.toList passedBachelorCoursesArray)

let passedMasterCoursesArray = inputArray.[13].Split   [|','|]
```

```
    let passedMasterCourses = set (Array.toList passedMasterCoursesArray)

    (studyType,studyName,studyLength,pointDist,startSeason,threeWeekDist,(cwscs,cwhcs,
        cuhcs),
     (scuhcs,scuscs),semestersAbroad,passedBachelorCourses,passedMasterCourses)
```

D.2 Generering af studieplaner

```
let generateStudyPlan inputFile outputFile =
    let xml = File.ReadAllText( __SOURCE_DIRECTORY__ + "\DTUCDB_Full.xml
        ")

    let (studyType, studyName, studyLength, ectsDistribution, semesterStartInSpring,
         threeWeekPeriodConstraints, courseConstraints,
         scheduleCodeConstraints, studyAbroadSemesters, passedBachelorCourses,
              passedMasterCourses) = readInput (inputFile + ".txt")

    let studyLevel = Map.find studyName studyLevelDatabase
    let studyCourses =
        match studyLevel with
        | 1 -> let (scs,_,_,_,_) = Map.find studyName bachelorStudyDatabase
               scs
        | _ -> let (scs,_,_) = Map.find studyName masterStudyDatabase
               scs

    let (courseWantedSoftConstraints,courseWantedHardConstraints,
         courseUnwantedHardConstraints) = courseConstraints
    let unifiedSet = Set.unionMany [courseWantedSoftConstraints;
         courseWantedHardConstraints; courseUnwantedHardConstraints]
    // We unify all courses from the course constraints with the other sets of courses.
         This gives us all relevant courses for the studyplan
    let courseSet = Set.unionMany (seq [(Set.map (fun (cn,_) -> cn) unifiedSet);
         studyCourses; passedBachelorCourses; passedMasterCourses])

    let (inputCourseDatabase, courseNumberMap) = readXmlFile xml courseSet

    // Implemented but not used
    let fieldRequirements = [|||]

    let solution =
        match studyType with
        | Partial -> solvePartial inputCourseDatabase courseConstraints
             scheduleCodeConstraints ectsDistribution threeWeekPeriodConstraints
             studyLength
                      fieldRequirements studyName passedBachelorCourses
                           passedMasterCourses semesterStartInSpring courseNumberMap
                           studyAbroadSemesters
        | Valid -> match studyLevel with
                   | 1 -> solveValidBachelor inputCourseDatabase courseConstraints
                        scheduleCodeConstraints ectsDistribution
                        threeWeekPeriodConstraints
                             studyLength fieldRequirements studyName
                                  passedBachelorCourses semesterStartInSpring
```

```
                        courseNumberMap studyAbroadSemesters
                | _ -> solveValidMaster inputCourseDatabase courseConstraints
                    scheduleCodeConstraints ectsDistribution
                    threeWeekPeriodConstraints
                        studyLength fieldRequirements studyName
                            passedBachelorCourses passedMasterCourses
                            semesterStartInSpring courseNumberMap
                        studyAbroadSemesters

let studyPlan =
    match solution with
    | Some (s,_) -> s
    | _ -> printfn "No solution"
           [|||]


let writer = new StreamWriter(( __SOURCE_DIRECTORY__ + "\\" + outputFile
      + ".txt"), false, Encoding.ASCII)
ignore [for i in 0..(( Array.length studyPlan) - 1) ->
            writer.WriteLine("Semester " + (Convert.ToString (i + 1)) + ":")
            let pointsum = (Set.fold (fun acc ((number:string,name,point,_,code,_,_,
                _),_,_,_,_) ->
                                    let realnumber = number.Substring(0,5)
                                    let theCourse = realnumber + " - " + name +
                                        " - " + (codeToString code)
                                    writer.WriteLine(theCourse)
                                    acc + point) 0.0 studyPlan.[i])
            writer.WriteLine("ECTS point: " + Convert.ToString(pointsum) + "\n")
        ]
writer.WriteLine(String.replicate 53 "-")
writer.WriteLine("Denne studieplan er genereret ud fra følgende input:")
let reader = new StreamReader(( __SOURCE_DIRECTORY__ + "\\" + inputFile
      + ".txt"),Encoding.ASCII)
writer.WriteLine("Studieplanstype              : " + reader.ReadLine())
writer.WriteLine("Studieretning                : " + reader.ReadLine())
writer.WriteLine("Studieplanslængde            : " + reader.ReadLine())
writer.WriteLine("Pointfordeling               : " + reader.ReadLine())
writer.WriteLine("Studiestart                  : " + reader.ReadLine())
writer.WriteLine("3-ugers fordeling            : " + reader.ReadLine())
writer.WriteLine("Kurser der skal vælges       : " + reader.ReadLine())
writer.WriteLine("Kurser der ønskes valgt      : " + reader.ReadLine())
writer.WriteLine("Kurser der ikke må vælges    : " + reader.ReadLine())
writer.WriteLine("Frie skemaplaceringer        : " + reader.ReadLine())
writer.WriteLine("Ønsket frie skemaplaceringer : " + reader.ReadLine())
writer.WriteLine("Udlandsophold                : " + reader.ReadLine())
writer.WriteLine("Beståede bachelor-kurser     : " + reader.ReadLine())
writer.WriteLine("Beståede kandidat-kurser     : " + reader.ReadLine())
writer.Close()

studyPlan
```

Appendiks E

Test

Indhold

E.1 Eksterne test

E.1.1 Test af input

K1Test.txt

```
Gyldig
Softwareteknologi
6
20-30,30-40,30,25-35,30,30
Forår
ja,ja,ja,nej,ja,nej
Kursusnummer1-0,Kursusnummer2-1
Kursusnummer3-0
Kursusnummer4-2
E1-1,F2-2
E3-3
30-4
Kursusnummer5
Kursusnummer6
EOF
```

Funktionskald

```
> readInput "K1Test.txt";;
val it :
  DataTypes.StudyType * string * int * (float * float) [] *
  bool * bool [] *
  (Set<string * int> * Set<string * int> * Set<string * int>) *
  (Set<DataTypes.ScheduleCode * int> *
   Set<DataTypes.ScheduleCode * int>) *
  Set<float * int> * Set<string> * Set<string> =
  (Valid, "Softwareteknologi", 6,
   [|(20.0, 30.0); (30.0, 40.0); (30.0, 30.0); (25.0, 35.0);
     (30.0, 30.0); (30.0, 30.0)|], true, [|true; true; true;
     false; true; false|],
   (set [("Kursusnummer3", 0)],
    set [("Kursusnummer1", 0); ("Kursusnummer2", 1)],
    set [("Kursusnummer4", 2)]),
   (set [(E1, 1); (F2, 2)], set [(E3, 3)]),
   set [(30.0, 4)], set ["Kursusnummer5"],
   set ["Kursusnummer6"])
```

E.1.2 Test af gyldig studieplan

```
Semester 1:
02213 - Praktisk digital konstruktion - January
02242 - Program analyse - E2A
02264 - Kravspecifikation - E3
42490 - Teknologi, Økonomi, ledelse og organisation - E5
ECTS point: 30

Semester 2:
02213 - Praktisk digital konstruktion - June
02220 - Distribuerede systemer - F1B
02233 - Netværkssikkerhed - F4A
02265 - Avancerede emner i software engineering  - F5B
02281 - Data-logik - F2B
02291 - System integration - F5A
ECTS point: 30

Semester 3:
ECTS point: 0

Semester 4:
Mas30 - Speciale - June
Mas30 - Speciale - F
ECTS point: 30

-----------------------------------------------------
Denne studieplan er genereret ud fra følgende input:
Studieplanstype              : Gyldig
Studieretning                : Informationsteknologi
Studieplanslængde            : 4
Pointfordeling               : 30,30,30,30
Studiestart                  : Efterår
3-ugers fordeling            : ja,nej,nej,ja
Kurser der skal vælges       : Speciale-4,02281-0
Kurser der ønskes valgt      : 42490-0
Kurser der ikke må vælges    : 02249-0
Frie skemaplaceringer        : E1A-1
Ønsket frie skemaplaceringer : F1A-2
Udlandsophold                : 30-3
Beståede bachelor-kurser     : 02156
Beståede kandidat-kurser     : 02257
```

E.1.3 Test af partiel studieplan

Partiel studieplan

```
Semester 1:
01005 - Matematik 1 - E5
01005 - Matematik 1 - E3B
02101 - Indledende programmering - E3A
31385 - Autonome robotsystemer - January
42101 - Introduktion til operationsanalyse - E2A
ECTS point: 23,75

Semester 2:
01005 - Matematik 1 - F5
01005 - Matematik 1 - F3B
01227 - Grafteori - F1B
02105 - Algoritmer og datastrukturer 1 - F2B
42430 - Projektledelse - F1A
ECTS point: 23,75

-----------------------------------------------------
Denne studieplan er genereret ud fra følgende input:
Studieplanstype              : Partiel
Studieretning                : Softwareteknologi
Studieplanslængde            : 2
Pointfordeling               : 20-30,20-30
Studiestart                  : Efterår
3-ugers fordeling            : ja,nej
Kurser der skal vælges       : 02101-0,02105-2
Kurser der ønskes valgt      : 02157-0
Kurser der ikke må vælges    : Bachelor-1,Bachelor-2
Frie skemaplaceringer        : E1A-1
Ønsket frie skemaplaceringer : F1A-2
Udlandsophold                :
Beståede bachelor-kurser     :
Beståede kandidat-kurser     :
```

Udvidelse af partiel studieplan

```
Semester 1:
01008 - Matematik 1 - temaøvelser - E
01017 - Diskret matematik  - E2B
01035 - Matematik 2 - E1A
02121 - Introduktion til softwareteknologi - E4B
```

```
02121 - Introduktion til softwareteknologi - January
02155 - Computerarkitektur - E2A
02156 - Logiske Systemer og Logikprogrammering - E4A
10020 - Fysik 1 - E1B
ECTS point: 36,25

Semester 2:
01008 - Matematik 1 - temaøvelser - F
02122 - Fagprojekt - Bachelor i Softwareteknologi - F5B
02122 - Fagprojekt - Bachelor i Softwareteknologi - June
02161 - Software Engineering 1 - F2A
02402 - Introduktion til statistik  - F4A
02405 - Sandsynlighedsregning - F4B
10020 - Fysik 1 - F1B
42610 - Ingeniørfagets videnskabsteori - F3A
ECTS point: 36,25

Semester 3:
02131 - Indlejrede systemer - E5
02159 - Operativsystemer - E1A
10044 - Fysik 2 - E4A
26027 - Grundlæggende kemi - E2A
27002 - Biovidenskab - E1B
ECTS point: 30

Semester 4:
02141 - Datalogisk modellering - F4
Bac20 - Bachelor - June
Bac20 - Bachelor - F
ECTS point: 30

--------------------------------------------------
Denne studieplan er genereret ud fra følgende input:
Studieplanstype              : Gyldig
Studieretning                : Softwareteknologi
Studieplanslængde            : 4
Pointfordeling               : 0-40,0-40,0-40,0-40
Studiestart                  : Efterår
3-ugers fordeling            : nej,nej,nej,nej
Kurser der skal vælges       :
Kurser der ønskes valgt      :
Kurser der ikke må vælges    :
Frie skemaplaceringer        :
Ønsket frie skemaplaceringer :
```

```
Udlandsophold                :
Beståede bachelor-kurser     : 01005,02101,31385,42101,
                               01227,02105,42430
Beståede kandidat-kurser     :
```

E.2 Interne test

E.2.1 Kildekode for interne test

```
let ctx = new Context()

// Functions to test

/// No point blocked courses are taken
let pointBlock (courseVariables:IntExpr []) (courseDatabase:CourseDatabase) (
    courseToIndex:CourseToIndex) (passedCourses:PassedCourses)
            (courseNumberMap:CourseNumberMap) =
    let assertPointBlock (courseNumber:CourseNumber) =
        let index = Map.find courseNumber courseToIndex
        ctx.MkAnd(ctx.MkEq(courseVariables.[index], ctx.MkInt 0), ctx.MkNot(ctx.MkBool
            (Set.contains courseNumber passedCourses)))
    let boolArray =
        [| for i in 0..((Array.length courseDatabase) - 1) ->
            let (_,_,_,_,_,pb,_,_) = courseDatabase.[i]
            let pbAlt = Set.fold (fun acc cn -> let set = Map.tryFind cn
                courseNumberMap
                                                match set with
                                                | Some s -> Set.union s acc
                                                | _ -> acc) Set.empty pb
            let boolArray = Set.toArray (Set.map (fun x -> assertPointBlock x) pbAlt
                )
            ctx.MkITE(ctx.MkGt(courseVariables.[i], ctx.MkInt 0), ctx.MkAnd(
                boolArray), ctx.MkBool true) :?> BoolExpr
        |]
    ctx.MkAnd(boolArray)

/// The studyplan has the right amount of semesters
let rightAmountOfSemesters (courseVariables:IntExpr []) (studyLength:StudyLength) (
    semesterStartInSpring:SemesterStartInSpring) =
    let boolArray =
        [| for c in courseVariables ->
            if semesterStartInSpring
            then ctx.MkOr(ctx.MkEq(c, ctx.MkInt 0),
                        ctx.MkAnd(ctx.MkLe(c, ctx.MkInt ((numberOfScheduleCodes
                            / 2) * (studyLength + 1))),
                                ctx.MkGe(c, ctx.MkInt ((numberOfScheduleCodes /
                                    2) + 1))))
```

```
                else ctx.MkAnd(ctx.MkLe(c, ctx.MkInt ((numberOfScheduleCodes / 2) *
                    studyLength)), ctx.MkGe(c, ctx.MkInt 0))
        |]
    ctx.MkAnd(boolArray)

/// Generates extended database
let extendDB (inputCourseDatabase:InputCourseDatabase) =
    /// Split a course
    let rec splitCourse (index : int) (numberOfScheduleCodes:int) (pcps:
     PartialCoursePairs) (courseDBList:List<Course>) (inputCourse:InputCourse) =
        let (courseNumber, courseName, ects, courseLevel, scs, pointBlock, mcps, fields)
            = inputCourse
        let scheduleCode = Set.minElement scs
        match Set.count scs with
        | 1 -> ((courseNumber, courseName, (ects / (float) numberOfScheduleCodes),
            courseLevel, scheduleCode, pointBlock, mcps, fields) :: courseDBList,
                Set.add (index, scheduleCode) pcps, index + 1)
        | _ -> let newDBList = (courseNumber, courseName, (ects / (float)
            numberOfScheduleCodes), courseLevel, scheduleCode, Set.empty, Set.empty,
            Set.empty) :: courseDBList
               let newPcps = Set.add (index, scheduleCode) pcps
               splitCourse (index + 1) numberOfScheduleCodes newPcps newDBList
                           (courseNumber, courseName, ects, courseLevel, Set.remove
                               scheduleCode scs, pointBlock, mcps, fields)

    let rec makeLists (i : int) (index : int) (courseDbList : List<Course>) (pcpss : Set
     <PartialCoursePairs>) =
        if i = Array.length inputCourseDatabase
        then (courseDbList, pcpss)
        else let inputCourse = inputCourseDatabase.[i]
             let (_, _, _, _, scs, _, _, _) = inputCourse
             let numberOfScheduleCodes = Set.count scs
             if numberOfScheduleCodes = 1
             then let (newDbList, _, newIndex) = splitCourse index
                      numberOfScheduleCodes Set.empty courseDbList inputCourse
                  makeLists (i + 1) newIndex newDbList pcpss
             else let (newDbList, newPcps, newIndex) = splitCourse index
                      numberOfScheduleCodes Set.empty courseDbList inputCourse
                  makeLists (i + 1) newIndex newDbList (Set.add newPcps pcpss)

    let (courseDbList, pcpss) = makeLists 0 0 [] Set.empty
    let courseDatabase = List.toArray (List.rev courseDbList)
    (courseDatabase, pcpss)


// General input
let testDatabase : CourseDatabase =
    [|("No1", "Kursus1", 5.0, 1, E1B, set["No2"], set [], set []) ;
      ("No2", "Kursus2", 15.0, 1, F3, set["No1"], set [], set []) ;
      ("No3", "Kursus3", 7.5, 2, E1, set [], set [], set []) ;
      ("No4", "Kursus4", 5.0, 1, F2B, set["No5"], set [], set []) ;
      ("No5", "Kursus5", 7.5, 2, F5A, set["No4"], set [], set []) ;|]

let testInputDatabase : InputCourseDatabase=
    [|("No1", "Kursus1", 5.0, 1, set[E1B], set["No2"], set [], set []) ;
```

```
        ("No2", "Kursus2", 15.0, 1, set [F3A;F4A;F3B], set["No1"], set [], set []) ;
        ("No3", "Kursus3", 7.5, 2, set [E1], set [], set [], set []) ;
        ("No4", "Kursus4", 5.0, 1, set [F2B], set ["No5"], set [], set []) ;
        ("No5", "Kursus5", 7.5, 2, set [F5A;F4B], set["No4"], set [], set []) ;|]

let courseToIndex = Map.ofList [("No1",0); ("No2",1); ("No3",2); ("No4",3); ("No5",4);]

let passedCourses = set["No1"]

let courseNumberMap = Map.ofList [("No1",set["No1"]); ("No2",set["No2"]); ("No3",set["
    No3"]); ("No4",set["No4"]); ("No5",set["No5"])]

let studyLength = 2

let semesterStartInSpring = false



// Point Block Test
let courseVariablesPointBlock1 =
    [|ctx.MkInt 1 :> IntExpr; ctx.MkInt 5 :> IntExpr; ctx.MkInt 7 :> IntExpr; ctx.MkInt
        2 :> IntExpr; ctx.MkInt 10:> IntExpr|]

let courseVariablesPointBlock2 =
    [|ctx.MkInt 0 :> IntExpr; ctx.MkInt 0 :> IntExpr; ctx.MkInt 4 :> IntExpr; ctx.MkInt
        0 :> IntExpr; ctx.MkInt 0:> IntExpr|]

let courseVariablesPointBlock3 =
    [|ctx.MkInt 0 :> IntExpr; ctx.MkInt 0 :> IntExpr; ctx.MkInt 0 :> IntExpr; ctx.MkInt
        0 :> IntExpr; ctx.MkInt 0:> IntExpr|]

let courseVariablesPointBlock4 =
    [|ctx.MkInt 0 :> IntExpr; ctx.MkInt 3 :> IntExpr; ctx.MkInt 0 :> IntExpr; ctx.MkInt
        9 :> IntExpr; ctx.MkInt 0:> IntExpr|]

pointBlock courseVariablesPointBlock1 testDatabase courseToIndex passedCourses
    courseNumberMap
pointBlock courseVariablesPointBlock2 testDatabase courseToIndex passedCourses
    courseNumberMap
pointBlock courseVariablesPointBlock3 testDatabase courseToIndex passedCourses
    courseNumberMap
pointBlock courseVariablesPointBlock4 testDatabase courseToIndex passedCourses
    courseNumberMap


// Right Amount Of Semesters Test
let courseVariablesRightAmountOfSemesters1 =
    [|ctx.MkInt 0 :> IntExpr; ctx.MkInt 5 :> IntExpr; ctx.MkInt 0 :> IntExpr; ctx.MkInt
        18 :> IntExpr; ctx.MkInt 35:> IntExpr|]

let courseVariablesRightAmountOfSemesters2 =
    [|ctx.MkInt 55 :> IntExpr; ctx.MkInt 78 :> IntExpr; ctx.MkInt 35 :> IntExpr; ctx.
        MkInt 42 :> IntExpr; ctx.MkInt 65:> IntExpr|]

let courseVariablesRightAmountOfSemesters3 =
```

```
    [|ctx.MkInt 1 :> IntExpr; ctx.MkInt 5 :> IntExpr; ctx.MkInt 7 :> IntExpr; ctx.MkInt
        2 :> IntExpr; ctx.MkInt 10:> IntExpr|]

let courseVariablesRightAmountOfSemesters4 =
    [|ctx.MkInt 1 :> IntExpr; ctx.MkInt 5 :> IntExpr; ctx.MkInt 28 :> IntExpr; ctx.MkInt
        18 :> IntExpr; ctx.MkInt 34:> IntExpr|]

rightAmountOfSemesters courseVariablesRightAmountOfSemesters1 studyLength
    semesterStartInSpring
rightAmountOfSemesters courseVariablesRightAmountOfSemesters2 studyLength
    semesterStartInSpring
rightAmountOfSemesters courseVariablesRightAmountOfSemesters3 studyLength
    semesterStartInSpring
rightAmountOfSemesters courseVariablesRightAmountOfSemesters4 studyLength
    semesterStartInSpring

// Extend Database Test
extendDB testInputDatabase
```

E.2.2 Constraint-funktioner

E.2.2.1 Pointspærring

Output 1:

```
val it : BoolExpr =
  (and (ite (> 1 0) (and (= 5 0) (not false)) true)
     (ite (> 5 0) (and (= 1 0) (not true)) true)
     (ite (> 7 0) and true)
     (ite (> 2 0) (and (= 10 0) (not false)) true)
     (ite (> 10 0) (and (= 2 0) (not false)) true))
```

Output 2:

```
val it : BoolExpr =
  (let ((a!1 (ite (> 0 0) (and (= 0 0) (not false)) true)))
  (and a!1
       (ite (> 0 0) (and (= 0 0) (not true)) true)
       (ite (> 4 0) and true)
       a!1
       a!1))
```

Output 3:

```
val it : BoolExpr =
  (let ((a!1 (ite (> 0 0) (and (= 0 0) (not false)) true)))
```

```
  (and a!1
       (ite (> 0 0) (and (= 0 0) (not true)) true)
       (ite (> 0 0) and true)
       a!1
       a!1))
```

Output 4:

```
val it : BoolExpr =
  (and (ite (> 0 0) (and (= 3 0) (not false)) true)
     (ite (> 3 0) (and (= 0 0) (not true)) true)
     (ite (> 0 0) and true)
     (ite (> 9 0) (and (= 0 0) (not false)) true)
     (ite (> 0 0) (and (= 9 0) (not false)) true))
```

E.2.2.2 Antal semestre

Output 1:

```
val it : BoolExpr =
  (and (and (<= 0 34) (>= 0 0))
     (<= 5 34)
     (>= 5 0)
     (and (<= 0 34) (>= 0 0))
     (<= 18 34)
     (>= 18 0)
     (<= 35 34)
     (>= 35 0))
```

Output 2:

```
val it : BoolExpr =
  (and (<= 55 34)
     (>= 55 0)
     (<= 78 34)
     (>= 78 0)
     (<= 35 34)
     (>= 35 0)
     (<= 42 34)
     (>= 42 0)
     (<= 65 34)
     (>= 65 0))
```

Output 3:

```
val it : BoolExpr =
  (and (<= 1 34)
     (>= 1 0)
     (<= 5 34)
     (>= 5 0)
     (<= 7 34)
     (>= 7 0)
     (<= 2 34)
     (>= 2 0)
     (<= 10 34)
     (>= 10 0))
```

Output 4:

```
val it : BoolExpr =
  (and (<= 1 34)
     (>= 1 0)
     (<= 5 34)
     (>= 5 0)
     (<= 28 34)
     (>= 28 0)
     (<= 18 34)
     (>= 18 0)
     (<= 34 34)
     (>= 34 0))
```

E.2.3 Opdeling af kurser med flere skemaplaceringer

Output:

```
val it : Course [] * Set<PartialCoursePairs> =
  ([|("No1", "Kursus1", 5.0, 1, E1B, set ["No2"],
      set [], set []);
     ("No2", "Kursus2", 5.0, 1, F3A, set [],
      set [], set []);
     ("No2", "Kursus2", 5.0, 1, F4A, set [],
      set [], set []);
     ("No2", "Kursus2", 5.0, 1, F3B, set ["No1"],
      set [], set []);
     ("No3", "Kursus3", 7.5, 2, E1, set [],
      set [], set []);
```

```
    ("No4", "Kursus4", 5.0, 1, F2B, set ["No5"],
     set [], set []);
    ("No5", "Kursus5", 3.75, 2, F5A, set [],
     set [], set []);
    ("No5", "Kursus5", 3.75, 2, F4B, set ["No4"],
     set [], set [])|],
  set [set [(1, F3A); (2, F4A); (3, F3B)];
       set [(6, F5A); (7, F4B)]])
```