KU LEUVEN

FACULTEIT INGENIEURSWETENSCHAPPEN

The ISP Course Selection Puzzle

Herbert Gorissen

Thesis voorgedragen tot het behalen van de graad van Master of Science in de ingenieurswetenschappen: elektrotechniek, optie Elektronica en geïntegreerde schakelingen

Promotor:
Prof. Gerda Janssens

Begeleider:
Matthias van der Hallen

Academiejaar 2015 – 2016

Preface

Herbert Gorissen

Contents

Abstract

Bij de start van een opleiding aan de KU Leuven is iedere toekomstige student verplicht zijn of haar opleiding samen te stellen uit een hele waaier aan opleidingsonderdelen. De selectie van opleidingsonderdelen is echter onderworpen aan een set van regels die allemaal voldaan moeten zijn wil men een geldig individueel studieprogramma (ISP) bekomen. Het ISP maakt deel uit van een specifieke groep van problemen genaamd configuratie problemen.

Het IDP systeem ontwikkeld aan de KU Leuven laat toe domein specifieke kennis uit te drukken in FO($\cdot$), een uitbreiding op eerste orde logica, ook wel de theorie genoemd. Eens de kennis beschreven is kan men met het bijhorende IDP systeem verscheidene vormen van inferentie toepassen op de theorie.

IDP als kennis representatie systeem is ideaal voor het beschrijven van en later redeneren over configuratie problemen. In het verleden heeft men er al grote configuratie problemen uit de bedrijfswereld succesvol mee kunnen beschrijven en oplossen. Interessant is om te achterhalen of we dit voor het ISP selectie probleem ook kunnen doen.

Momenteel zijn er bij het opstellen van het ISP ook een aantal tekortkomingen die we met IDP zouden willen oplossen. Zo wordt het lessenrooster niet mee in rekening gebracht, waardoor studenten niet weten of ze opleidingsonderdelen opnemen waarvan de lessen mogelijk kunnen overlappen. En wat als er een foutieve selectie wordt gemaakt? Momenteel krijgt de student na het bevestigen van diens selectie een foutmelding en deze is vaak nog onduidelijk. IDP voorziet momenteel een aantal functies om de oorzaken van inconsistentie op te sporen at run-time. Handig zou zijn om hiermee korter op de bal te spelen en meteen bij een foutieve selectie uitleg te kunnen geven waarom de voorlopige huidige selectie niet correct is en hoe men dit kan oplossen.

Naast dit alles heeft deze thesis nog een andere grote doelstelling. Het opsporen van oorzaken van inconsistenties ofwel conflict explanation genoemd is een gebied waar momenteel nog veel onderzoek gaande is. Enkele interessante technieken zijn hier al voorgelegd en het is in onze interesse om na te gaan of en hoe we deze kunnen integreren in het IDP systeem.

List of Figures and Tables

List of Figures

List of Tables

List of Abbreviations and Symbols

Abbreviations

LoG	Laplacian-of-Gaussian
MSE	Mean Square error
PSNR	Peak Signal-to-Noise ratio

Symbols

42	"The Answer to the Ultimate Question of Life, the Universe, and Everything" according to
c	Speed of light
E	Energy
m	Mass
π	The number pi

Chapter 1

Introduction

1.0.1 Constraint Satisfaction Problems

CSP's zijn problemen waar voor een reeks variabelen een geldige waarde uit het domein dient toegekend te worden. De toekenning echter is gebonden aan een set van regels waaraan de variabelen moeten voldoen. Formeel definiëren we een CSP als een triple $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$, met $\mathcal{X}$ de verzameling van variabelen, $\mathcal{D}$ de domeinwaarden voor deze variabelen en $\mathcal{C}$ de constraints over de variabelen. Typisch worden dit soort problemen opgelost d.m.v. zoeken door alle mogelijk combinaties, deze recursieve methode gaat voor elke nieuwe toekenning na of de constraints voor de partiële toekenning consistent zijn. Zo ja, dan volgt er een nieuwe recursieve oproep, anders gaat het algoritme backtracken. Het oplossen van CSP's met een eindig domein behoort tot de klasse van NP-Complete problemen en de berekeningen zijn vaak van hoge complexiteit met betrekking tot de grootte van het domein. Om het zoekproces proberen te versnellen bestaan er variaties op backtracking zoals backmarking en backjumping. De eerste zorgt voor een efficiëntere manier om consistentie na te gaan en backjumping is een betere manier van backtracking waarbij men over meerdere waarden tegelijk kan backtracken. Deze technieken versnellen het zoekproces in dat ze onnodige stappen detecteren en overslaan. Anderzijds bestaan er propagatie technieken die lokale consistentie garanderen met technieken zoals arc consistency, hyper-arc consistency en path-consistency die de domeinen van de variabelen proberen te verkleinen alvorens het zoekproces gestart wordt.

1.0.2 Interactive Configuration Problems

In deze thesis ligt de focus op een specifieke groep van problemen binnen het domein van CSP's genaamd Interactieve Configuratie (IC) problemen. Constraint Satisfaction Problems zijn problemen waar een toekenning van domeinwaarden voor de variabelen dient gezocht te worden waarvoor geldt dat aan alle constraints voldaan is. IC problemen hebben hetzelfde doel, maar bij CSP's wordt er naar een toekenning gezocht d.m.v. recursieve zoektechnieken terwijl het in het geval can IC problemen een gebruiker is die handmatig stap per stap een geldige toekenning gaat proberen samenstellen. Om de gebruiker hierin bij te staan is er nood aan software, software

die de gebruiker zo goed mogelijk bijstaat gedurende dit hele proces. Uit ervaring is gebleken dat het schrijven van software voor dit soort problemen geen gemakkelijke opgave is. Zo is aangetoond dat een imperatieve aanpak voor het beschrijven van de regels van een probleem vaak moeilijk is (Gelle and Weigel, 1996). Reden hiervoor is dat de regels over de betreffende domeinkennis vaak verspreid zit in de software in codesnippets. Daarboven is onderhoud van zulke software een enorm moeilijke opgave waarbij de kleinste wijziging in de constraints kan leiden tot een volledige herwerking van de code. Een alternatief hiervoor is een daclaratieve aanpak. In declaratieve toepassingen kunnen de regels van een probleem beschreven worden d.m.v. logische constraints. Gelle en Weigel Gelle and Weigel (1996) laten niet alleen zien dat de declaratieve bescvhrijving van een probleem overzichtelijker en duidelijker is maar ook dat een wijziging in de constraints gemakkelijk aangepast kan worden. En hoewel er met een imperatieve aanpak optimalere algoritmen ontwikkeld kunnen worden, is aangetoond dat declaratieve methoden ook goede prestaties kunnen neerzetten (**?**).

1.0.3 Knowledge Representation: Het IDP Systeem

Er ondertussen vele declaratieve systemen verschenen (Prolog, Ant, Lisp, ...). Wat opvalt is dat elk systeem vaak samenhangt met één enkele specifieke vorm van inferentie. Dus afhankelijk van de gewenste inferentie zal een probleem opnieuw moeten beschreven worden in een ander systeem dat deze inferentie kan toepassen, ondanks dat de kennis telkens dezelfde is. Deze gedachtegang ligt aan de basis van concept van Knowledge Representation (Denecker and Vennekens, 2008). In het KR-paradigma wordt gesteld dat ongeacht de inferentie los staat van de kennis over een probleem. Deze kennis is niets meer dan een verzameling van informatie, maar met deze informatie kunnen meerdere vormen van inferentie toegepast worden. In dit onderzoek ligt de aandacht op IDP, een KR-systeem ontwikkeld aan de K.U. Leuven en voor het eerst voorgesteld in 2008. Het laat toe om de kennis over een probleem te beschrijven in FO($\cdot$), dit is eerste orde logica maar uitgebreid met aggregaten, type definities en inductieve definities.

1.1 Probleemstelling

1.1.1 Individueel Studieprogramma

Een ISP samentstellen valt onder deze categorie van Interactieve Configuratieproblemen. De gebruiker, in dit geval een toekomstige student wil een geldig ISP bekomen d.m.v. het selecteren van mogelijke opleidingsonderdelen. Maar de student kan niet zomaar elk opleidingsonderdeel selecteren, er zijn een heleboel regels waaraan de selectie moet voldoen. Deze regels zijn op zichzelf duidelijk en intuïtief, maar om een selectie vinden die aan alle regels voldoet kan mogelijk verwarrend zijn. Het huidige systeem voorziet weinig ondersteuning in het bijstaan van de gebruiker gedurende het selectieproces. Het is pas als je een selectie bevestigt dat het systeem controleert of deze ook effectief correct is volgens de regels. En hoewel het systeem wel weergeeft aan welke regels er (in geval van inconsistentie) niet voldaan is, moet de gebruiker

zelf op zoek gaan naar de selectie die verantwoordelijk is verantwoordelijk is hiervoor. De regels van een geldig ISP nemen geen lessenrooster in acht, wat dus betekent dat lesmomenten voor verschillende opleidingsonderdelen kunnen overlappen.

1.2 Doel

In deze thesis wil ik onderzoeken of de regels van het ISP efficiënt kunnen beschreven worden in FO($\cdot$). De regels zijn voor elke opleiding anders. De vraag is of het mogelijk is om met één enkele theorie in IDP alle opleidingen correct te kunnen beschrijven. Belangrijk hierbij is dat de domeinen sterk kunnen verschillen tussen opleidingen, en dat de eventuele theorie hier mee om moet kunnen gaan.

De K.U. Leuven heeft haar eigen systeem voor het samenstellen van het ISP. Een minpunt van dit systeem is de ondersteuning die het biedt tijdens het selectieproces. /*WAT ZIJN DE TEKORTKOMINGEN*/ Hier kan beter gedaan worden, daarom wil ik meer ondersteuning proberen bieden. In een zelf ontworpen Front-end met grafische user interface wil ik de volgende functionaliteiten integreren:

Automatisch invullen van gevolgen Als de student een vak A kiest en hieruit volgt dat vak B ook gevolgt moet worden, dan is het de bedoeling dat het systeem dit automatisch invult zodat de student zich hier verder niets van hoeft aan te trekken.

Detectie van foutieve selectie Deze functionaliteit is momenteel al aanwezig in het huidige systeem. Maar de detectie gebeurt pas bij de bevestiging van de selectie i.p.v. op het moment van de selectie zelf. En dat is wat ik zal proberen te integreren in de nieuwe front-end. En niet enkel dit, maar ook het effectief opsporen van de oorzaak zodat de gebruiker dit kan aanpassen.

Geldig ISP laten genereren Stel dat een student keuzes heeft gemaakt omtrend de vakken die hij/zij echt wil of niet wil volgen, maar de selectie is nog geen volledige oplossing. Dan kan de student vragen aan het systeem om de selectie verder in te vullen.

Optimaal ISP laten genereren Niet alleen moet het systeem een geldige oplossing kunnen genereren, maar ook de beste oplossing volgens een bepaalde criterium. Zo zou een student bijvoorbeeld graag een ISP willen waarbij de werklast zo goed mogelijk verdeeld is over beide semester.

Ongedaan maken van acties Deze functionaliteit is terug te vinden in zowat de meeste moderne systemen. Als een student ontevreden is over zijn/haar recente keuzes, moet er de mogelijk zijn om deze ongedaan te kunnen maken.

Weergave van het lessenrooster Voorheen had ik al vermeld dat studenten geen duidelijk overzicht hebben van het lessenrooster dat voortvloeit uit hun keuzes. Vakken kunnen lesmomenten hebben die mogelijk overlappen met die van andere vakken en dit kan voor een ongewenste verrassing zorgen eens het ISP

bevestigd is. Het is mijn bedoeling om in de nieuwe front-end wel een eventueel lessenrooster weer te geven.

De meeste van deze functionaliteiten steunen op inferentie technieken die IDP aanbeidt zoals model expansie, minimizatie, propagatie etc. Dit wil zeggen dat om deze functionaliteiten te kunnen aanbieden, de onderliggende inferentie technieken efficiënt moeten werken. Herinner dat we te maken hebben met een IC probleem en dit vereist dat de reactietijd niet meer dan enkele seconden bedraagt. Ik zal dus moeten onderzoeken of inferentie goed verloopt binnen dit probleem.

Het opsporen van foutieve selecties is één ding, maar kunnen verklaren wat er mis is met de selectie is een ander verhaal. Het domein van Conflict explanation bevat nog een heel aantal openstaande vragen. IDP zelf voorziet zelf de unsatcore, en methode die opzoek gaat naar de regels die (in geval van unsatisfiability) nooit waar gemaakt kunnen worden. Probleem is echter dat de output geformuleerd is in FO(·), en een doorsnee gebruiker zal niet in staat zijn om hieruit iets te kunnen afleiden. Verscheidene andere technieken voor conflict explanation die zijn voorgelegd leggen de focus op het optimaliseren van de rekentijd (**?**) of het zoeken naar minimale correcties (**?**). Maar in verklaring geven in een formaat dat begrijpbaar is voor iedereen, daarvoor bestaat nog geen algemene techniek. En in deze thesis zet ik een stap in het ongewisse om op zoek te gaan naar mogelijke oplossingen.

1.3 Dependencies

FO(·) Het is de bedoeling om de mogelijkheden van IDP in actie te zien en met een proof of concept de prestaties van het systeem te testen. FO(·) bezit een grote uitdrukkingskracht, bovenop eerste order logica bevat het ook taalelementen zoals aggregaten (sum, avg, min, max, ..) en inductieve definities.

Kivy Bij de keuze van de programmeertaal en framework viel mijn oog op Kivy. Dit is een Python library met een hele waaier aan grafische elementen die elk uitgebreid geconfigureert kunnen worden. Hiervoor kan gebruik gemaakt worden van een door de uitgever ontwikkelde kv-language. Kivy applicaties werken cross-platform en de applicaties zijn event-driven gebruik makend van een centrale loop.

JSON Het gegevensformaat van JSON is ideaal om de domeinen van verschillende opleidingen in weer te geven. De domeinwaarden zitten gekoppeld aan een attribuut, dit maakt het gemakkelijk om structuur te parsen en zo het domein voor iederen opleiding in te lezen.

Chapter 2

Implementation

2.0.1 Preliminaries

De kennis over het ISP staat beschreven in een theorie T over een vocabularium Σ. Dit vocabularium bestaat uit een set Σ_p van predicaatsymbolen en een set van functiesymbolen Σ_f. Een interpretatie I bestaat uit D de verzameling domeinelementen, een mapping van elk functiesymbool f/n naar een functie met ariteit n op D en een mapping van elk predicaatsymbool P/n naar een relatie $R \subseteq D^n$. Een three-valued partiële interpretatie bestaat uit een mapping op drie waarheidswaarden u,t,f respectievelijk onbekend, waar en niet waar. De waarheidswaarden zijn partieel geordend volgens precisie, $u \leq_p f$ en $u \leq_p t$. De bedoeling is om een exacte interpetatie te bekomen met maximale precisie die consistent is met de theorie T.

2.1 Knowledge base constructie

Geen opleiding aan de KU Leuven is dezelfde, de opleidingsonderdelen verschillen net zoals de vakgroepen waar ze deel van uitmaken. Voor sommige vakgroepen ben je verplicht alle onderdelen op te nemen, terwijl je voor andere en minimaal (of zelfs maximaal) aantal studiepunten ervan moet opnemen. Kortom de regels binnen éénzelfde opleiding zijn niet zo moeilijk om te beschrijven, maar om dit te veralgemenen en een theorie te creëren die voor alle opleidingen een geldig ISP kan beschrijven dat is een andere paar mouwen. Voor het gemak van de lezer noemen we opleidingsonderdelen vanaf nu vakken. Hieronder staan de verschillende onderdelen van de knowledge base beschreven.

2.1.1 Vocabulary

Om te beginnen zal ik de vocabulair toelichten, beginnend bij de types waarna ik de functies en predicaten zal beschrijven.

Vak De naam verklaart zichzelf al, dit type omvat de verzameling van de verschillende vakken in het domein. De domeinwaarden zijn niet de naam in natuurlijke taal, maar de unieke vakcode.

VakNaam Elk vak heeft een naam geschreven in natuurlijke taal oftwel de vaknaam.

VakGroep Vakgroepen zijn een heel algemeen concept, het omvat elke verzameling van vakken waarvoor bepaalde regels gelden. Per groep gelden er andere regels die uniek zijn voor die groep, maar ook het type waartoe de groep behoort.

VakGroepType Elke vakgroep maakt deel uit van een bepaald type groep, en per type groep gelden er bepaalde regels bovenop de regels van de individuele groep zelf.

Fase Opleidingen bestaan uit fases oftewel jaren, de Master Computer Wetenschappen bestaat bijvoorbeeld uit 2 fases. Vakken binnen een opleiding kunnen enkel gevolgd worden tijdens de fase van de opleiding waartoe ze behoren.

Semester De K.U. Leuven werkt met een semester systeem, dit wil zeggen dat elke fase (schooljaar) is opgedeeld in 2 semesters. Elk vak kan gevolgd worden in de semester waartoe het behoort. Er bestaan ook jaarvakken waarvoor de werklast verdeeld is tussen beide semesters, een voorbeeld hiervan is de Masterproef.

Studiepunten Elk vak bevat studiepunten, deze beschrijven de geschatte werklast voor dit vak. Vakgroepen vereisen vaak dat je een bepaalde hoeveelheid een studiepunten opneemt.

IsType(VakGroep,VakGroepType)

Verplicht(Vak,VakGroep)

InVakGroep(Vak,VakGroep)

InFase(Vak,Fase)

Geselecteerd(Vak,Fase)

NietGeselecteerd(Vak)

HeeftNaam(Vak):VakNaam

MinAantalStudiepunten(VakGroep):Studiepunten

MaxAantalStudiepunten(VakGroep):Studiepunten

AantalStudiepunten(Vak):Studiepunten

GeselecteerdAantalStudiepuntenPerVakGroep(Vakgroep):Studiepunten

GeselecteerdAantalStudiepuntenPerSemester(VakGroep):Studiepunten

InSemester(Vak):Semester

Vakgroeptypes

Zo valt op dat we telkens dezelfde **types** van vakgroepen tegenkomen. Deze types die steeds terugkomen in meerdere opleidingen hebben ongeacht de opleiding waar ze in voorkomen dezelfde regels die ermee gepaard gaan. Zo is elke opleiding op zich een vakgroep van het type opleiding, het bevat vakken die je verplicht bent te volgen en keuzevakken. Het heeft een minimum (en maximum) aantal studiepunten dat de student verplicht moet opnemen. Een ander belangrijk type vakgroep is de (hoofd)specialisatie, verscheidene opleidingen geven de keuze tussen meerdere specialisaties. Verwacht wordt dat je van minstens 1 zo'n specialisatie alle verplichte vakken opneemt. Naast hoofdspecialisatie bestaat er ook het type verdere specialisatie, waarin de student verplicht wordt een bepaald aantal studiepunten op te nemen aan vakken uit deze of bepaalde andere vakgroepen. Vervolgens is er het type 'Algemeen vormende en onderzoeksondersteunende groep' vakgroep, dit is veruit het moeilijkste type groep om te beschrijven. De student moet opnieuw een bepaald aantal studiepunten aan vakken opnemen uit deze groep. Maar daarnaast gelden er ook specifieke regels voor verscheidene vakken die deel uitmaken ervan. En als laatse is er het type 'Bachelor verbredend pakket', waarin studenten die beginnen aan hun masteropleiding vakken moeten opnemen die ontbraken in hun bacheloropleiding.

Deze types zien we het vaakst voorkomen en kunnen ongeacht de opleiding met dezelfde set van regels beschreven worden. Het is mogelijk dat sommige types van vakgroepen niet telkens voorkomen, en dus de regels ook niet van kracht zijn. In een bacheloropleiding zal bijvoorbeeld nooit een vakgroep voorkomen van het type Bachelor verbrendend pakket.

2.1.2 Theorie van het lessenrooster

2.2 Front-End

Herinner dat het samenstellen van het ISP een Interactief configuratieprobleem is, en om aan de eigenschappen van zo'n IC probleem te kunnen voldoen moet de reactietijd van de inferentiemethoden snel genoeg zijn. Om dit te testen heb ik een front-end applicatie ontwikkeld die met behulp van een grafische interface een student toelaat een ISP samen te stellen. Deze input wordt vertaald naar FO($\cdot$) en doorgespeeld naar IDP. Vervolgens voert IDP inferentie uit en de resultaten hiervan worden terug doorgespeeld naar de front-end, die het resultaat zal weergeven d.m.v. de GUI. Zo kunnen de prestaties van IDP in een gebruiksvriendelijke omgeving getest worden.

2.2.1 Grafische Gebruikersinterface

2.2.2 Communicatie tussen Front- en Back-end

Niet alleen is de syntax tussen IDP en Python compleet verschillend, het zijn ook twee verschillende klassen van programmeertalen respectievelijk declaratief en objectgeoriënteerd. Dus communicatie tussen de twee talen is niet vanzelfsprekend. Een Python API die toelaat IDP te gebruiken is reeds ontwikkeld (Vennekens, 2015).

Het doel van de API is om de kloof tussen de declaratieve omgeving van IDP en Python kleiner te maken. Door een populaire programmeertaal te kiezen hoopt de auteur dat informatici met een achtergrond in declaratieve talen sneller van IDP gebruik zullen maken hoewel ze de syntax van IDP niet meester zijn. Persoonlijk heb ik wel al ervaring met IDP en heb dus geopteerd om geen gebruik te maken van de API.

De input/output van IDP is compleet tekstueel, de front-end moet een volledig tekst-bestand genereren in een formaat dat IDP kan begrijpen en verwerken. De output van IDP is opnieuw een tekst-bestand dat de front-end zal moeten parsen om het resultaat te kunnen verwerken. Zo'n tekst-bestand heeft altijd dezelfde opmaak, het bevat een theorie, vocabulair, structuur, procedure en eventueel termen in geval van minimizatie. De front-end bevat een parser object dat voor elke oproep zo'n tekst bestand zal opbouwen. De theorie, vocabularium en de procedures zijn altijd dezelfde en hoeven slecht éénmalig ingelezen te worden. Enkel de structuur zal bij elke nieuwe actie gedeeltelijk dynamische gegenereerd moeten worden. De predicaten in het vocabularium zijn opgedeeld in twee verzamelingen. De eerste verzameling Γ bevat de predicaten die vooraf gegeven zijn en gekend zijn door het systeem. Anders gezegd bestaat er een interpretatie G voor Γ. De andere verzameling Ω, zijn de predicaten waar nog geen interpretatie W voor gegeven is en die door de gebruiker (in dit geval de student) ingevuld zullen moeten worden. Tot deze laatste groep behoren volgende predicaten: Geselecteerd(Vak,Fase) en NietGeinteresseerd(Vak). De bedoeling is om een invulling te zoeken zodat $W \cup G$ consistent is met alle regels van de theorie T.

2.3 Features

2.3.1 Selectieproces

Elke keuze die de gebruiker maakt moet zorgvuldig afgehandeld worden om te kunnen geranderen dat de selectie altijd satisfieerbaar is. De voorwaarde is dat als de gebruiker een keuze maakt die ervoor zorgt dat de theorie niet langer satisfieerbaar is, hij hier meteen op de hoogte van wordt gebracht zodanig dat hij dit kan oplossen alvorens verder te gaan. Er wordt gestart met de minst precieze interpretatie W_0 om uiteindelijk stapsgewijs tot een exacte interpretatie W_n met maximale precisie te komen waarvoor geldt dat $W_n \cup G \models T$. Hoe deze stappen verlopen staat beschreven in onderstaand algoritme. Hier wordt W opnieuw opgedeeld in drie verzamelingen U, P en O. U is de verzameling van predicaten met een waarheidswaarde $\geq_P u$, waarvoor geldt dat deze toegekend zijn door de gebruiker. De tweede verzameling P bevat eveneens predicaten met een waarheidswaarde $\geq_p u$, maar deze zet zijn de propagaties die volgen uit U. O tenslotte is de set van predicaten waarvoor de waarheidswaarde nog onbekend is. Voor W_0 geldt: $W_0 = O_0$ en $U_0 = P_0 = \emptyset$. In het verder verloop van de tekst zal deze beschrijven blijven gelden en terugkomen.

Algorithm 1: Selectieproces

```
1  function Selectieproces (U_i);
   Input  : De nieuwe keuze van de gebruiker P_i
   Output :
2  if Sat(U_i) then
3  |  V ← P_i-1 ∩ O_i;
4  |  if V ≠ ∅ then
5  |  |  V' ← KeuzeGebruiker(V);
6  |  |  Selectieproces(U_i ∪ V');
7  |  else
8  |  |  P_i ← Propagate(U_i);
9  |  |  NieuweGebruikerActie(U_i-1,U_i, P_i-1,P_i, O_i-1,O_i);
10 |  end
11 else
12 |  N ← Unsat(U_i);
13 |  K,L ← ManueleResolutie(N);
14 |  U'_i ← U_i [K] - L;
15 |  Selectieproces(U'_i);
16 end
```

2.3.2 Propagatie

Bij het samenstellen van een ISP kan het systeem de informatie die de gebruiker reeds verder gaan propageren. Hierbij baseert het systeem zich op de regels van de theorie. Neem het volgende voorbeeld:

$$\mathrm{Geselecteerd}(\mathrm{A}) \Rightarrow \mathrm{Geselecteerd}(\mathrm{B})$$

Deze regel zegt dat als A geselecteerd is, B ook geselecteerd moet zijn. Als de gebruiker A selecteerd, zal het systeem hieruit afleiden dat B ook geselecteerd moet doen. In de GUI zal te zien zijn dat B ook geselecteerd is, hoewel de gebruiker dit niet expliciet heeft gekozen.

2.3.3 Model Expansie

Als de gebruiker al zijn/haar voorkeuren heeft ingevuld in de partiële interpretatie W_i met $U_i \cup P_i \cup O_i = W_i$ en $O_i \neq \emptyset$. Dan kan het systeem de verzameling van predicaten O_i waarvan de waarheidswaarde nog onbekend is, verder oplossen. Om zo tot een interpretatie W_f to komen met $U_i \cup P_i \subseteq W_f$. W_f is tevens (precisie)maximaal en $W_f \cup G \models T$. Anders gezegd zal het systeem de partiële selectie verder vervolledigen tot een geldig ISP.

2.3.4 Minimizatie

Het is niet alleen mogelijk om een partiële selectie verder te laten vervolledigen, maar om hierbij ook een parameter in acht te nemen en een interpretatie W_f te bekomen waarbij deze parameter zo klein mogelijk is. Het systeem kan een optimalisatie van volgende parameters zoeken:

Werklast Met werklast wordt bedoeld het aantal studiepunten dat opgenomen wordt. Een student is verplicht vakken te selecteren en gekoppeld aan die vakken zijn studiepunten. De student kan vragen aan het systeem om een ISP samen te stellen waarbij de werklast (de som van de studiepunten van de geselecteerde vakken) ze laag mogelijk is.

Werklast per semester Vakken die geen jaarvakken zijn, vallen ofwel in het eerste semester of het tweede. Het systeem kan een ISP samenstellen waarbij het verschil in studiepunten tussen geselecteerde vakken van eerste en tweede semester zo klein mogelijk is. Zodat de werklast zo goed mogelijk verdeeld is tussen beide semesters.

Overlap Vakken hebben uiteraard lesmomenten en het kan al eens gebeuren dat deze lesmomenten voor verschillende vakken samenvallen. Dit is uiteraard niet ideaal voor de student om die niet op twee plaatsen tegelijk kan zijn. Stel dat er het lesmoment van vak A gepland is van 16u tot 18u en het lesmoment voor vak B vindt plaats op dezelfde dag van 15u tot 17u. De overlap bedraagt dan 60 minuten. De student kan een ISP laten genereren waarbij de totale som van alle overlap zo minimaal mogelijk is.

2.3.5 Ongedaan Maken

Het ongedaan kunnen maken van selecties is een van de aspecten waar meerdere strategiëen mogelijk zijn. Elke stap in het selectieproces (nieuwe selectie door de gebruiker) wordt bijgehouden in een zogenaamde actie. Deze actie bevat niet alleen die nieuwe interpretatie W_i maar ook de voorgaande W_{i-1} zodanig dat als en actie ongedaan gemaakt wordt het systeem weet wat de toestand voorheen was en hier naar kan terugkeren. Het is niet noodzakelijk zo dat in een actie A_i met interpretaties W_{i-1} en W_i dat $W_{i-1} \leq_p W_i$. Acties beschrijven elke handeling in de tijd gemaakt door de gebruiker, dus ook de waarheidswaarde van een predicaat minder specifiek maken. Ongedaan maken in deze context betekent dus niet automatisch de huidige selectie minder specifiek maken, maar eerder terugkeren in de tijd.

Strategie 1a De eerste strategie houdt in dat als een actie ongedaan gemaakt wordt de interpretatie W_i simpelweg vervangen wordt door de voorgaande W_{i-1}. Dit is wellicht de meest simpele strategie aangezien er geen extra bewerkingen uitgevoerd moeten worden buiten het terugkeren naar de voorgaande interpretatie.

Strategie 1b Bij de voorgaande strategie staat een actie ongedaan maken gelijk aan simpelweg de klok terugdraaien en W_i vervangen door W_{i-1}. Dus eventuele

Algorithm 2: MaakActieOngedaan

```
1 function MaakActieOngedaan (Ac);
  Input  : De actie die ongedaan gemaakt moet worden A_i
  Output :
2 verwijder(A_i);
3 W_i ← W_i-1;
```

propagaties die volgden uit de keuze van de gebruiker in de actie worden ook ongedaan gemaakt. En in plaats van alles simpelweg ongedaan te maken zou het beter zijn als de gebruiker de keuze krijgt om eventuele propagaties te behouden. Dit zorgt echter voor het volgende probleem. De assumptie is dat acties handelingen weergeven in de tijd gemaakt door de gebruiker, het ongedaan maken van handelingen wordt dus gezien als het terugdraaien van de tijd. Maar als de gebruiker kiest om een actie ongedaan te maken en tegelijk eventuele propagaties toch wil behouden dan geldt deze assumptie niet meer. In **??** staat beschreven hoe dit probleem opgelost kan worden. Het is simpelweg een uitbreiding op de eerste strategie, een actie wordt nog altijd verwijderd en de laatste interpretatie W_i wordt vervangen door de voorgaande W_{i-1}. Maar waar het hier voorheen stopte gebeurt er nu het volgend. Er wordt gecontroleerd of er een set van predicaten V bestaat die zowel toebehoren aan O_{i-1} en P_i. Ofwel predicaten waarvoor de waarheidswaarde eerst onbekend was en daarna een meer specifieke waarheidswaarde hebben gekregen d.m.v. propagatie als gevolg van de keuze van de gebruiker. Is deze set niet leeg dan zal de gebruiker moeten kiezen in KeuzeGebruiker welke van de propagaties te houden. In het geval dat de gebruik kiest om propagaties te behouden, worden deze niet langer gezien als propagaties maar eerder als keuzes gemaakt door de gebruiker. Deze worden toegevoegd een de verzameling U en dit wordt gezien als een nieuwe handeling in het selectieproces waarvoor een nieuwe actie gecreëerd zal worden.

Algorithm 3: MaakActieOngedaan

```
1  function MaakActieOngedaan (Ac);
   Input  : De actie die ongedaan gemaakt moet worden A_i
   Output :
2  V ← O_i-1 ∩ P_i;
3  verwijder(A_i);
4  W_i ← W_i-1;
5  if V ≠ ∅ then
6  |  V' ← KeuzeGebruiker(V);
7  |  if V' ≠ ∅ then
8  |  |  Selectieproces(U_i-1 ∪ V');
9  |  end
10 end
```

Strategie 2 Voorheen was de assumptie dat een actie ongedaan maken gelijk stond aan de tijd als het ware terugdraaien en de interpretatie vervangen door de voorgaande. En dat dit niets te maken had met de precisie ordening van de interpretaties. Voor deze strategie wordt er niet meer uitgegaan van deze assumptie, maar in plaats daarvan staat ongedaan maken gelijk aan eender welke keuze van de gebruiker die ervoor zorgt dat de interpretatie minder precies wordt. Dit kan zijn de waarheidswaarde van een predicaat ongedaan (minder precies) maken of een waarheidswaarde van een predicaat veranderen van bv. waar naar onwaar. Met als gevolg dat de interpretatie minder precies is omdat propagaties niet meer gelden. Wat verandert is dat in deze situatie geen expliciete acties meer worden bijgehouden zoals voorheen, en dat ongedaan maken nu een onderdeel is van het selectieproces zelf. Kort gezegd verandert enkel de assumptie over ongedaan maken. De rest van de werking blijft dezelfde, behalve dat zoals voorheen niet meer gebruikt worden. De werking staat beschreven in 4.

Algorithm 4: Selectieproces

```
1  function Selectieproces (U_i);
   Input  : De nieuwe keuze van de gebruiker P_i
   Output :
2  if Sat(U_i) then
3  |  V ← P_{i-1} ∩ O_i;
4  |  if V ≠ ∅ then
5  |  |  V' ← KeuzeGebruiker(V);
6  |  |  Selectieproces(U_i ∪ V');
7  |  else
8  |  |  P_i ← Propagate(U_i);
9  |  |  ~~NieuweGebruikerActie(U_{i-1},U_i,P_{i-1},P_i, O_{i-1},O_i);~~
10 |  end
11 else
12 |  N ← Unsat(U_i);
13 |  K,L ← ManueleResolutie(N);
14 |  U'_i ← U_i [K] - L;
15 |  Selectieproces(U'_i);
16 end
```

2.4 Conflict Explanation

Het is mogelijk dat de gebruiker een verkeerde keuze maakt waardoor de theorie niet meer satisfieerbaar is. Simpel gezegd heeft de gebruiker een waarde gekozen voor een bepaalde variabele, zodanig dat dit ervoor zorgt dat één of meerdere regels uit de theorie samen niet meer waar kunnen worden gemaakt.

$$\mathrm{Geselecteerd}(A) \wedge \mathrm{Geselecteerd}(B)$$

Het bovenstaande voorbeeld van een theorie bevat ëën regel die zegt dat A en B beide geselecteerd moeten zijn. Stel dat de gebruiker nu kiest ¬ Geselecteerd(A). Hieruit volgt dat de regel uit de theorie nooit waar kan zijn, of niet satisfieerbaar wordt. Dit is een conflixt en het is aan het systeem om een de gebruiker duidelijk te maken dat er een probleem is, wat het probleem inhoudt en hoe de gebruiker dit kan oplossing. Dit laatste behoort tot het domein van conflict explanation. IDP gebruikt momenteel twee technieken om oorzaken van niet satisfieerbaarheid op te sporen. De unsatstructure spoort een set van variabelen op die het probleem veroorzaken. Vervolgens is er de unsattheory, die zoekt naar een minimale set van regels uit de theorie die niet waar gemaakt kunnen worden gegeven de huidige selectie.

Unsatstructure

De unsatstructure is een efficiënte tool om aan de gebruiker te kunnen meedelen, welke van de voormalige selecties problemen veroorzaken. In het voorbeeld van het ISP werd er een opsplitsing gemaakt tussen twee sets van predicaten, Γ waarvoor de invulling G vooraf bekend is en Ω waarvoor een interpretatie W samengesteld dient te worden door de student. W bestond zelf opnieuw uit drie verzamelingen U, P en O. Als een interpretatie ervoor zorgt dat de theorie niet meer satisfieerbaar wordt, dan is dit te wijten aan de keuze van de gebruiker U. En het is van deze verzameling dat de unsatstructure een precisie minimale deelverzameling V zal zoeken als oorzaak van het probleem. De student krijgt een oplijsting te zien van V, en zal de mogelijkheid krijgen om hier veranderingen in aan te brengen zodoende het probleem op te lossen.

Reified Constraints

Wat opvalt is dat de voorgaande techniek enkel een verzameling selecties teruggeeft die bijdragen tot het probleem. Maar verdere uitleg over welke regel(s) uit de theorie niet meer waar gemaakt kunnen worden en waarom wordt niet gegeven. Dus de gebruiker krijgt enkel de foute selectie te zien, zonder erbij te zeggen wat er juist mis mee is. Herinner de unsattheory zoekt achter en minimale set van regels uit de theorie die niet meer waar gemaakt kunnen worden. Dit is het exact hetgene wat de gebruiker hoort te weten om het probleem op te lossen, maar het probleem zit het hem in de formulering. De unsattheory formuleert de fouten in de IDP syntax FO(·). Deze syntax is voor de doorsnee informaticus met een achtergrond in eerste orde logica of IDP goed te lezen, maar voor een student die geen ervaring heeft met deze domeinen zal ongetwijfeld niet in staat zijn deze regels te ontcijferen. Neem als voorbeeld onderstaande regel, elke student computerwetenschappen zal vrij snel kunnen achterhalen wat de regel inhoud. Maar personen zonder deze achtergrond zullen dit niet kunnen verstaan.

Listing 2.1: IDP Rule Example

```
∀vg[VakGroep] : IsType(vg,AVO) ⇒ GesAantalStupunVakGr(vg)
= sum{v[Vak], sp[Studiepunten],f[Fase] : InVakGroep(v,vg)
∧ Geselecteerd(v,f) ∧ AantalStudiepunten(v)=sp : sp }.
```

Een mogelijke oplossing voor dit probleem komt in de vorm van reified constraints. De theorie T bestaat uit regels (constraints) en hierin wordt onderscheid gemaakt tussen 2 sets van regels. De eerste is de set van regels B die altijd waar zullen zijn, deze noemen we achtergrond constraints. De andere set is die van regels F genaamd voorgrond constraints die de gebruiker moet zien waar te maken d.m.v. een correcte interpretatie samen te stellen. Het is deze set van regels F, waarvoor reified constraints gebruikt zullen worden. Voor zij die niet bekend zijn met het concept van reified constraints, het is een zeer simpele techniek. Een regel C (constraint) kan heel simpel omgevormd worden tot een reified constraint door C equivalent te maken aan een booleaanse waarde B. Dus een regel C ziet er als dan als volgt uit, B $\leftrightarrow$ C. De waarheidswaarde van C is dan equivalent aan die van B, dus als de regel niet consistent of 'false' is, zal B dus ook 'false' zijn, hetzelfde geldt ook voor 'true' uiteraard. De voorgaande regel ziet er dan als volgt uit.

Listing 2.2: IDP Reified Constraint Example

```
Rule ↔ ∀vg[VakGroep] : IsType(vg,AVO) ⇒ GesAantalStupunVakGr(vg)
= sum{v[Vak], sp[Studiepunten],f[Fase] : InVakGroep(v,vg)
∧ Geselecteerd(v,f) ∧ AantalStudiepunten(v)=sp : sp }.
```

Rule is de booleaanse variabele wiens waarheidswaarde die van de regel reflecteert. Op deze manier hoeven we enkel na te gaan of Rule true of false is om te weten te komen of de regel (in)consistent is gegeven de selectie van de gebruiker. Dit alleen biedt natuurlijk niet veel extra informatie, maar het nut van de booleaanse variabele is wel degelijk belangrijk. Voor elke regel uit F is er op voorhand een beschrijving opgesteld in natuurlijke taal met de redenering achter het eventuele falen van die regel. Aan de hand van de reified constraints kan er heel gemakkelijk achterhaald worden welke regels er wel of niet consistent zijn door simpelweg te kijken naar de booleaanse waarde in de regels. Is de booleaanse variabele false, dan moet de gebruiker de bijhorende beschrijving te zien krijgen voor deze regel. Hoewel dit een zeer gemakkelijke en generische manier is om aan conflict explanation te doen, zijn de mogelijkheden van deze techniek toch gelimiteerd en niet altijd voldoende toereikend. Ten eerste is de uitleg vaak oppervlakkig en niet specifiek genoeg om duidelijk te kunnen verklaren waarom een regel inconsistent is.

Listing 2.3: Reified constraint Shortcomings

```
∀vg[VakGroep] : MaxAantalStudiepunten(vg) > 0 ⇒
MinAantalStudiepunten(vg) ≤ GeselecteerdAantalStudiepuntenPerVakGroep(vg)
```

Bovenstaande regel stelt dat voor elke vakgroep, het totaal aantal geselecteerde studiepunten van deze vakgroep tussen diens minimum en maximum moet liggen, als het maximum aantal studiepunten ervan groter is dan nul. In het geval dat deze regel inconsistent is krijgt de student de volgende uitleg te zien. Je hebt niet voor alle vakgroepen tussen het minimum en maximum aantal studiepunten geselecteerd. De verklaring is algemeen en geeft geen specifieke details. Zo zegt het niet voor welke vakgroep het geselecteerd aantal studiepunten niet klopt, en of het er nu teveel of te weinig zijn. Dus de verklaringen zijn redelijk beperkt in hun mogelijkheden.

En ten tweede laat deze methode enkel toe om inconsistente regels op te sporen. Een Interactief Configuratieprobleem zoals het ISP bevat een verzameling predicaten Ω waarvoor de gebruiker (in dit geval een student) stapsgewijs een interpretatie W samenstelt. Deze interpretatie moet uiteindelijk als ze compleet is, een geldig model zijn volgens de regels van de theorie. Een regel is inconsistent als ze niet voldaan is gegeven de huidige partië selectie. Dit wil niet zeggen dat de regel niet meer consistent kan worden naarmate de interpretatie verder ingevuld wordt door de student. Terwijl een set van regels die nooit samen waar gemaakt kunnen worden gegeven de huidige partiële selectie niet-satisfieerbaar zijn. Het is deze laatste set van regels die dient gevonden te worden. De techniek van reified constraints op zich kan dit niet verwezenlijken. Maar door deze techniek te combineren met de unsatstructure wordt dit wel een mogelijkheid.

Chapter 3

Evaluation

De bedoeling van dit onderzoek is om de regels van het ISP te kunnen beschrijven in IDP, zodanig dat ze voor elke opleiding aan de KU Leuven een correct model kunnen genereren. Daarnaast is het de bedoeling om het lessenrooster mee in rekening te brengen. Aangezien het mogelijk is dat vakken overlappen, is het toch belangrijk dat de student op de hoogte is van deze informatie. Met deze regels willen we natuurlijk inferentie kunnen doen, en een belangrijke vereiste van een Interactief configuratieprobleem zoals een ISP samenstellen is een snelle respons. De inferentietaken moeten dusdanig snel afgehandeld kunnen worden dat de reactietijd hooguit enkel seconden bedraagt. En tenslotte is er conflict explanation, waar bij een foute selectie de gebruiker een specifieke uitleg hoort te krijgen wat er mis is, waarom en hoe het opgelost kan worden. Dit alles in een zo natuurlijk mogelijke taal die de gebruiker gamakkelijk kan verstaan.

Theorie van het ISP Geen twee opleidingen zijn dezelfde, en toch is het belangrijk dat ongeacht deze verschillen we telkens dezelfde set van regels kunnen gebruiken zonder daarin te moeten gaan aanpassen. Om de omvang van het project haalbaar te houden voor één persoon, is het domein van opleidingen beperkt gebleven tot die van de computerwetenschappen en informatica. Hiervoor ben ik erin geslaagd en theorie op te stellen die zonder probleem eender welk van deze opleidingen kan beschrijven. Hoewel de naamgeving vaak verschilt per opleiding, komen toch vaak structeren voor die dezelfde eigenschappen vertonen. Dus de echte uitdaging is deze structuren vinden, niet de regels ervoor schrijven. Voor een klein sub-domein binnen de opleidingen van de KU Leuven is het dus mogelijk een theorie te vinden.

Het lessenrooster

Grafische Interface De GUI dient als prototype voor een mogelijke versie die de student kan gebruiken in de toekomst. Daarin moet het minstens de functionaliteit bieden van het huidig systeem, maar met een aantal verbeteringen. Eerst en vooral is het de bedoeling dat de student de verschillende inferentietechnieken van de IDP back-end kan aanroepen om hulp te bieden bij het

selectieproces. Het huidige systeem geeft enkel bij bevestiging terug of de selectie al dan niet correct is volgens de regels. De nieuwe GUI doet dit in real-time op het moment van de foutieve selectie en weergeeft de oorzaak van het probleem. Naast dit alles is het ook de bedoeling om in het nieuwe systeem het lessenrooster te betrekken. De student krijgt een weekoverzicht te zien van de lessen voor de vakken die hij/zij geselecteerd heeft. Zo kan er worden gecontroleerd of de lessen mogelijk overlappen. Daarbij is het ook mogelijk om IDP een ISP te laten samenstellen waarbij de lessen zo min mogelijk overlappen. De ontwikkeling van de GUI was zeer kostelijk, en heeft langer geduurd dan eerst gehoopt. De hoge configureerbaarheid van de elementen heeft tot gevolg dat het meer tijd kost om ze correct te implementeren. Hoewel de kv-language dit process vergemakkelijkt ligt de tijdsduur nog altijd hoger dan verwacht. Een van de belangrijkste criteria waar ik niet naar heb gekeken bij het kiezen van het framework is KIS (Keep It Simple). En dit zal ik zeker niet vergeten in de toekomst.

Inferentie Met behulp van de interface kan de gebruiker een ISP samenstellen. Het is de bedoeling om de gebruiker bij te staan in dit proces, dit door gevolgen van bepaalde keuzes door te voeren, een onvolledige selectie te vervolledigen, een optimaal ISP samen te stellen volgens een bepaald criterium, foute keuzes te detecteren en de gebruiker hiervan op de hoogte te brengen etc. En niet te vergeten, al deze processen moeten in real-time gebeuren en dus zeer snel afgehandeld kunnen worden. Binnen het domein van de opleidingen, ben ik tot de conclusie gekomen dat zowat al deze inferentie taken snel zeer snel en efficiënt uitgevoerd worden. De reactietijd, meegerekend het genereren van de IDP text file en het resultaat terug ontcijferen bedraagt in bijna alle gevallen minder dan 1 seconde. Zelfs de resultaten van minimizatie opdrachten geven goede resultaten terug met een maximale rekentijd van ongeveer 10 seconden.

Conflict Explanation Als de gebruiker een keuze maakt waardoor de regels onsatisfieerbaar worden, dan is het de bedoeling dat de oorzaak hiervan opgespoord wordt en de gebruiker hierover wordt ingelicht. Via de unsatstructure van IDP krijgt de gebruiker een lijst te zien met vakken waarvoor hij/zij een keuze heeft gemaakt die de onsatisfieerbaarheid veroorzaken. Het is in deze lijst dat de gebruiker keuzes kan veranderen of ongedaan maken om det probleem op te lossen. Enkel een lijst tonen is natuurlijk niet genoeg, als er geen uitleg gegeven wordt waarom de huidige keuze fout is kan je ook niet weten wat je moet veranderen om het terug op te lossen.

Reified constraint bieden extra informatie en dienen als een helpende hand om de gebruiker door het selectieprocess te loodsen. Deze simpele generische manier om inconsistente regels op te sporen en te verklaren in natuurlijke taal heeft zeker zijn voordelen. De hoeveelheid werk vereist om deze functionaliteit te implementeren is zeer weinig. Maar de functionaliteit van deze techniek is echter beperkt. Het laat toe om inconsistente regels op te sporen in tegenstelling tot wat we zoeken namelijk de oorzaken van onsatisfieerbaarheid. Daarbij is

de uitleg die voorzien wordt bij een inconsistente regel mogelijk niet toereikend voor de gebruiker, zo kan er niet aangewezen worden voor welke instantie van de variabelen een bepaalde regel inconsistent is, en het concept dat de regel beschrijft is vaak heel algemeen waardoor de gebruiker er moeilijk iets uit kan afleiden. Ondanks deze minpunten is deze techniek zeker nuttig in de zin dat het de gebruiker in de juiste richting kan leiden tijdens het selectieproces, en dit voor een lage implementatiekost.

Amilhastre paper

Chapter 4

Conclusion

Chapter 5

Related Work

Separating Knowledge from Computation: An FO(·) Knowledge Base System and its Model Expansion Inference ? Binnen het domein van declaratief programmeren, ziet men vaak de trend dat zo'n systeem een logische taal implementeert samen met een specifieke vorm van inferentie. Recent heeft men het paradigma Knowledge Base System of KBS voorgesteld met het idee dat kennis over een probleem niet gekoppeld hoeft te zijn aan één bepaalde vorm van inferentie. In plaats daarvan stelt men voor de kennis apart uit te drukken in een declaratieve taal waarop men dan verschillende vormen van inferentie kan toepassen. In deze paper wordt zo een systeem voorgesteld namelijk IDP. Het laat toe kennis over een bepaald probleem neer te schrijven in FO(·), oftewel eerste orde logica met een aantal uitbreidingen zoals types, partiële functies, aggregaten en inductieve definities. Om mezelf bekend te maken met IDP en het paradigma van kennis representatie, is dit de paper bij uitstek. Het beschrijft het oorspronkelijke doel van IDP en motiveert de keuzes die zijn gemaakt bij het ontwerpen ervan. De paper begint met een korte samenvatting van eerste orde logica, aggregaten en inductieve regels en vervolgens wordt uitgelegd hoe men deze regels uitdrukt in de syntax van het IDP systeem. De verschillende inferentie technieken die ondersteund zijn zoals model expansion, optimization, deduction, model checking enzovoort worden toegelicht. Vervolgens geeft de paper een blik achter de schermen en beschrijft het de moeilijkheden van inferentie en de oplossingen die hiervoor voorzien zijn. Dit is een intro voor iedereen die zich wil verdiepen in IDP, het geeft je de wat, waarom en hoe die je nodig hebt om eraan te beginnen. De syntax in de paper is echter ondertussen niet meer up-to-date.

Interactive Configurations and the KB-Paradigm ? Deze paper gaat na of interactieve configuratie problemen kunnen opgelost worden m.b.v. declaratieve methodes en meer specifiek een KR-systeem. Algemeen bekend is dat dit soort problemen enorm complex zijn wil men ze programmeren in een imperatieve programmeertaal **?**. De reden hiervoor is dat regels elkaar beïnvloeden en een simpele wijziging van ook maar één regel ervoor kan zorgen dat andere regels niet meer kloppen. De regels zijn ook vaak lang, onoverzichtelijk en slecht leesbaar, problemen die volgens de paper op te lossen zijn met IDP. Gebruik makend van FO(·) en de

syntax van het IDP systeem gaat men proberen de constraints van zo 'n probleem te beschrijven en er vervolgens over redeneren door er verschillende vormen van inferentie op toe te passen. Specifiek wordt er gekeken naar de complexiteit van een aantal verschillende vormen van inferentie die IDP aanbiedt. Geconcludeerd wordt dat ondanks de limitaties van een KR-systeem (grote structuren kunnen tijd van het berekenen opblazen) stelt men toch vast dat de aanpak voldoende goede resultaten boekt. Men kan dit zien als een proof of concept waarbij een echt configuratie probleem uit de bedrijfswereld gebruikt wordt als voorbeeld. Een ISP selecteren is in essentie een interactief configuratie probleem, dus de resultaten die voortvloeien uit deze paper kunnen als leidraad gezien worden voor deze thesis.

A Logical Framework for Configuration Software ? In deze paper neemt de auteur opnieuw de klasse van configuratie problemen onder de loep. In deze verzameling van problemen staat software de gebruiker bij in het maken van bepaalde keuzes volgens een aantal regels. Het schrijven van software hiervoor met imperatieve programmeertalen is vaak een lastige opgave, het is een naïeve manier van opstellen waarbij elke keuze een if-else regel vereist en de kleinste verandering ervoor kan zorgen dat de regels niet meer kloppen. Dit is voor programmeurs vaak een nachtmerrie als de regels van een systeem geregeld wijzigen (bv. Tax on web). De auteur stelt een declaratieve aanpak voor en specifiek met het oog gericht op het model van kennis representatie. Het doel is om bestaande problemen efficiënter te kunnen oplossen. Als voorbeeld heeft men gekozen voor het probleem waar een student wordt gevraagd zijn opleiding samen te stellen volgens de regels van de universiteit. De regels kunnen gemakkelijk beschreven worden in IDP, dat gebruik maakt van een uitgebreide versie van FO($\cdot$). De expressiviteit ervan wordt gezien als een groot pluspunt en maakt het heel gemakkelijk voor de programmeur om regels elegant en leesbaar te beschrijven. Er worden verscheidene vormen van inferentie toegepast in de paper waaronder model expansion, propagation, model checking en conflict explanation. Opvallend is dat al deze technieken werken met dezelfde domein logica, één van de kern ideeën en grote voordelen van het KR-systeem. Er wordt beschreven hoe elke techniek te werk gaat en uiteindelijk toont de paper aan dat voor het gekozen probleem de inferentie voldoende efficiënt en in polynomiale tijd verloopt. Hierbij komt ook nog dat de elegantie van een declaratieve aanpak, de grote expressiviteit van FO($\cdot$) en de efficiëntie van een kennis representatie systeem vele voordelen bieden in tegenstelling tot imperatieve technieken. De technieken die de paper beschrijft voor het oplossen van het gekozen configuratie probleem kunnen als basis dienen voor mijn verder werk en zijn uitermate geschikt om op verder te bouwen. De bedoeling is om technieken toe te voegen zoals minimization en de theorie verder uit te breiden met kennis over het lessenrooster.

Predicate Logic as a Modelling Language: The IDP System ? Recente doorbraken in automated reasoning technieken zoals SAT-solving and constraint programming zijn de aanleiding tot een groeiende interesse in logica als programmeertaal. Ondertussen zijn er al verscheidene declaratieve programmeertalen verschenen. Hier

stelt men IDP voor als kennis representatie systeem, waar domein logica volledig onafhankelijk is van de inferentie die men erop wil toepassen. Imperatieve talen laten toe een specifiek algoritme te schrijven dat hoge prestaties kan leveren. Hier tegenover staat dat kennis over het probleem verweven zit in de code, wat onderhoud enorm moeilijk of soms zelfs onmogelijk maakt. Vaak kunnen ze ook maar één taak uitvoeren, in tegenstelling tot IDP waar verschillende inferentie taken dezelfde domein logica gebruiken. De syntax in IDP is makkelijk te begrijpen en leunt dicht aan bij natuurlijke taal, iets wat bij imperatieve programmeertalen vaak niet het geval is. En tenslotte hoeft de programmeur in IDP zich geen zorgen te maken over hoe een probleem moet opgelost worden. Hij moet enkel de domein logica beschrijven en IDP doet de rest. Er zijn ook limitaties, imperatieve oplossingen zijn vaak sneller dan een algemene declaratieve aanpak. Toch worden de prestaties van IDP zeker voldoende geacht. De rekentijd is voldoende kort dat het kan gebruikt worden in een interactieve omgeving waarbij de gebruiker in minder dan enkele seconden een antwoord verwacht. Dit is een van de belangrijkste resultaten van de paper aangezien het aantoont dat IDP wel degelijk in staat is dit en andere gelijkaardige problemen met voldoende efficiëntie op te lossen. Naast de motivatie van de paper worden alle belangrijke aspecten van IDP uitgelegd. Zo wordt beschreven hoe de FO($\cdot$) in mekaar zit, waarna de paper toont hoe men dit in de syntax van IDP zelfs kan schrijven. Er wordt ook een motivatie gegeven waarom men de keuze van FO($\cdot$) heeft gemaakt, belangrijk hier zijn vooral de expressiviteit, begrijpbaar-heid en herbruikbaarheid. Ook wordt uitgelegd hoe IDP under the hood werkt met technieken zoals ground-and-bound, symmetry exploitation en functional dependency detection.

Consistency Restoration and Explanations in Dynamic CSP's: Application to Configuration ? In de klasse van interactieve configuratie problemen is het belangrijk dat de technieken snel en efficiënt werken. Bepaalde functionaliteiten zoals het voorzien van uitleg bij een foute keuze van de gebruiker en herstellen van consistentie zijn echter nog altijd moeilijk op te lossen en nemen teveel tijd in beslag om ze interactief te kunnen gebruiken. In deze paper proberen ze een oplossing te bieden voor deze problemen. Het voorstel is om een automaat te bouwen die alle verschillende mogelijke selecties voorstelt. De voorwaarde is wel dat deze automaat vooraf off-line wordt opgesteld, dit hoeft maar één maal te gebeuren. Gebruik makend van deze automaat toont de paper aan dat de voorheen vermelde technieken op een efficiënte en snelle manier kunnen berekend worden. Om hun worden kracht bij te zetten hebben ze een configuratie probleem van Renault voor het configureren van een auto gebruikt om hun model te testen. De resultaten ervan zijn trouwens zeer indrukwekkend. IDP voorziet momenteel wel een techniek voor het opsporen van conflicten. Maar de technieken voorgesteld in de paper zijn veelbelovend en het is zeker interessant om eens te kijken of we deze toch niet in IDP kunnen integreren.

Lowering the learning curve for declarative programming: a Python API for the IDP system ? Nog te vaak zijn programmeurs terughoudend als het aankomt op declaratief programmeren vanwege de leercurve die ermee gepaard gaat.

Het doel van de auteur is om de moeilijkheidsgraad te verlagen in de hoop dat het de stap naar declaratieve systemen vergemakkelijkt. De auteur probeert dit te bereiken door een IDP KR-systeem te integreren in een Python API. Python is één van de meest gebruikte programmeertalen, en dankzij de API kan iedereen vertrouwd met Python gebruik maken van het het IDP KR-systeem zonder de syntax te moeten leren. Doch zijn declaratieve systemen inherent verschillend van imperatieve systemen, en iedereen die geen opleiding omtrent declaratieve systemen heeft genoten zal dit dan toch nog moeten inhalen. Bij studenten Computerwetenschappen is dit echter wel het geval aangezien het standaard onderdeel is van de opleiding. En de auteur hoopt met de API dat studenten sneller de stap zullen maken naar IDP, door ze de mogelijkheid te geven om dit te doen in de vaak vertrouwde Python omgeving. De API is zeker een goede overbrugging, maar ik ben ondertussen bekend met de syntax van IDP. En de efficiëntie van de syntax en de goede leesbaarheid ervan hebben mij toch doen kiezen om geen gebruik te maken van de API.

Generating Corrective Explanations for Interactive Constraint Satisfaction ? In het veld van interactieve configuratie problemen is het zo dat als de gebruiker een ongeldige combinatie van variabelen selecteert we op zoek gaan naar de oorzaak hiervan. Anders gezegd gaan we een verklaring zoeken voor het probleem. Dit is natuurlijk handig voor de gebruiker om te achterhalen welke selectie hij moet aanpassen om het conflict op te lossen. Maar nog handiger zou zijn dat niet enkel de oorzaak wordt getoond, maar ook een (of meerdere) mogelijke oplossing(en), m.a.w. een selectie die zo veel mogelijk dezelfde is als die van de gebruiker, maar die wel tot een mogelijk juiste oplossing kan leiden. De paper stelt CORRECTIVEEXP voor, een systematisch algoritme voor het zoeken van minimal corrective explanations. Een vergelijkende studie met reeds bestaande technieken op verscheidene grote problemen toont aan dat het algoritme zeer goede resultaten boekt. Dit is essentieel aangezien dat interactieve configuratie problemen een snelle reactietijd vereisen.

5.0.1 Huidige stand van zaken

Eerste punt op de agenda was de literatuurstudie, IDP en het KR-paradigma waren voor mij een volkomen nieuw concept. De papers die hierboven beschreven staan hebben veel duidelijkheid gebracht in een voor mij compleet onbekende omgeving. Eens wegwijs in het domein van IDP, was de volgende stap het leren van de syntax en een eerste poging ondernemen om de regels van het ISP voor te stellen in IDP. Hierop volgden vele iteraties om het ontwerp van het ISP te verbeteren. Momenteel is er een werkende versie, maar sommige vormen van inferentie en specifiek minimisatie nemen veel tijd in beslag. Het is van belang dat deze regels in de theorie opnieuw bekeken en aangepast dienen te worden. Verwacht wordt dat de gebruiker een gebruiksvriendelijke en simpele GUI krijgt om mee te werken. Deze is volop in ontwikkeling en de eerste vormen van functionaliteit zijn al voorzien. Momenteel kan een gebruiker keuzes maken en een ISP laten genereren. De mogelijkheid om een optimaal ISP te laten generen is voorzien, maar aangezien de minimisatie in IDP te veel rekentijd in beslag neemt is het afgeraden hier momenteel gebruik van te

maken. Eveneens worden keuzes gemaakt door de gebruiker al gepropageerd. De GUI is geschreven in Python waarbij er gebruik is gemaakt van de Kivy library. Kivy voorziet een hele waaier aan classes voor het samenstellen van een GUI en heeft zelfs zijn eigen Kivy taal voor het creëren van een GUI en de functionaliteit ervan te beschrijven.

Bibliography

Marc Denecker and Joost Vennekens. Building a knowledge base system for an integration of logic programming and classical logic. In Logic Programming, pages 71–76. Springer, 2008.

Esther Gelle and Rainer Weigel. Interactive configuration using constraint satisfaction techniques. In PACT-96, pages 37–44, 1996.

Joost Vennekens. Lowering the learning curve for declarative programming: a python api for the idp system. arXiv preprint arXiv:1511.00916, 2015.

Fiche masterproef

Student: Herbert Gorissen

Titel: The ISP Course Selection Puzzle

Nederlandse titel: The ISP Course Selection Puzzle

UDC: 621.3

Korte inhoud:
Here comes a very short abstract, containing no more than 500 words. LaTeX commands can be used here. Blank lines (or the command `\par`) are not allowed!

Thesis voorgedragen tot het behalen van de graad van Master of Science in de ingenieurswetenschappen: elektrotechniek, optie Elektronica en geïntegreerde schakelingen

Promotor: Prof. Gerda Janssens

Assessor:

Begeleider: Matthias van der Hallen