



## **Het vereenvoudigen van FOBDD's met behulp van Presburger aritmetiek**

door

Pieter VAN HERTUM

Promotor: prof. dr. M. Denecker

Begeleiders: Bart Bogaerts, Stef De Pooter

Proefschrift ingediend tot het  
behalen van de graad van  
Master in de Wiskunde

Academiejaar 2011-2012

© Copyright by KU Leuven

*Zonder voorafgaande schriftelijke toestemming van zowel de promotor(en) als de auteur(s) is overnemen, kopiëren, gebruiken of realiseren van deze uitgave of gedeelten ervan verboden. Voor aanvragen tot of informatie i.v.m. het overnemen en/of gebruik en/of realisatie van gedeelten uit deze publicatie, wend u tot de KU Leuven, Faculteit Wetenschappen, Geel Huis, Kasteelpark Arenberg 11, 3001 Leuven (Heverlee), Telefoon +32 16 32 14 01.*

*Voorafgaande schriftelijke toestemming van de promotor(en) is eveneens vereist voor het aanwenden van de in dit afstudeerwerk beschreven (originele) methoden, producten, schakelingen en programma's voor industrieel of commercieel nut en voor de inzending van deze publicatie ter deelname aan wetenschappelijke prijzen of wedstrijden.*

# Voorwoord

Voorwoord

# Inhoudsopgave

Voorwoord	ii
Inhoudsopgave	iv
Lijst van figuren	v
Lijst van Algoritmen	vi
Inleiding	1
<b>1 Voorbeschouwing</b>	<b>3</b>
1.1 Elementaire begrippen . . . . .	3
1.1.1 Propositielogica . . . . .	3
1.1.2 Eerste-ordeloga . . . . .	7
1.1.3 Uitbreidingen op eerste-ordeloga . . . . .	10
1.2 Kennisrepresentatie . . . . .	11
1.2.1 Voorbeelden . . . . .	13
<b>2 Binary Decision Diagrams</b>	<b>17</b>
2.1 BDDs in propositielogica . . . . .	17
2.1.1 Binary Decision Trees . . . . .	17
2.1.2 Van BDT naar BDD . . . . .	19
2.2 BDDs voor Predicatenlogica . . . . .	21
2.2.1 FOBDDs . . . . .	21
2.2.2 Opstellen van FOBDDs in de praktijk . . . . .	23
<b>3 Kennisrepresentatie en het IDP-systeem</b>	<b>27</b>
3.1 Modelgeneratie en -expansie . . . . .	27
3.1.1 Werking van modelexpansie . . . . .	27
3.1.2 IDP . . . . .	29

3.1.3	Grounden . . . . .	30
3.2	Verbeteringen voor een groundingsalgoritme . . . . .	32
3.2.1	Grounden met grenzen . . . . .	32
3.2.2	Equivalente subformules herkennen . . . . .	37
3.2.3	Vereenvoudigen van FOBDDs . . . . .	38
3.2.4	Aritmetische redundantie . . . . .	40
<b>4</b>	<b>Vereenvoudiging van aritmetische FOBDDs</b>	<b>43</b>
4.1	Inleiding . . . . .	43
4.2	Een eerste versie van het algoritme . . . . .	45
4.3	Het beslissen van de redundantie . . . . .	46
4.3.1	Presburger aritmetiek . . . . .	46
4.3.2	Beslisbaarheid . . . . .	50
4.3.3	Beslissen van Presburger redundantie in de praktijk . . . . .	59
4.4	Uiteindelijk algoritme . . . . .	60
4.5	Voorbeeld . . . . .	60
4.6	Uitbreidingen . . . . .	63
4.7	Implementatie . . . . .	69
4.7.1	Het toevoegen van formules aan een Omega+ verzameling . . . . .	70
4.7.2	Aanpassingen voor FOBDDs . . . . .	71
4.7.3	Aanpassingen voor de Bruijn-indices . . . . .	73
4.7.4	Het vereenvoudigingsalgoritme . . . . .	73
4.8	Resultaten bij Experimenten . . . . .	74
<b>5</b>	<b>Besluit</b>	<b>77</b>
5.1	Theorie . . . . .	77
5.2	Algoritme . . . . .	78
5.3	Implementatie . . . . .	78
5.4	Verder onderzoek . . . . .	79
<b>A</b>	<b>Getypeerde eerste-ordeloga</b>	<b>82</b>
<b>B</b>	<b>Populariserende samenvatting</b>	<b>84</b>

# Lijst van figuren

1.1	Het battleshipprobleem . . . . .	14
1.2	Een Sudoku-puzzel . . . . .	16
2.1	Het opbouwen van een BDT voor $x \wedge (y \vee z)$ . . . . .	18
2.2	Het delen en reduceren van BDT's . . . . .	19
2.3	EFOBDD voor $\exists x(P(x) \wedge \forall y(P(y)))$ . . . . .	22
2.4	Het Bottom-Up opbouwen van een FOBDD voor $x = 2 \wedge \exists a : (a+1 < x \vee P(a))$ . . . . .	26
3.1	Valse afhankelijkheid in een FOBDD . . . . .	39
3.2	Het gebruik van de Bruijn-indices in een FOBDD . . . . .	40
3.3	Aritmetische redundante in een FOBDD voor $\{r, c \mid \forall x(Water(x, c) \vee 1 > x - r \vee r < x)\}$ . . . . .	41
3.4	Aritmetische redundante in een FOBDD voor $\{y[\mathbb{N}] \mid (y = 1 \vee y = 2) \wedge (y \neq 2 \vee y \neq 3)\}$ . . . . .	42
4.1	De padformuleverzamelingen van alle knopen in een FOBDDT . . . . .	45
4.2	FOBDDTs . . . . .	64
4.3	Aritmetische redundantie bij bepaalde volgordes . . . . .	65
4.4	FOBDDTs bij bewijs van Stelling 4.2 . . . . .	67
4.5	Presburger redundantie in getypeerde FOBDDTs . . . . .	69

# Lijst van algoritmen

1.1	Tseitin Transformatie . . . . .	6
3.1	Volledige Grounding . . . . .	31
3.2	Grounding met grenzen van een zin $\varphi$ . . . . .	34
4.1	simpBDT: aritmetische vereenvoudiging van FOBDTs . . . . .	47
4.2	simpBDT1: aritmetische vereenvoudiging van FOBDTs (tweede versie) . . .	61
4.3	simpBDT2: aritmetische vereenvoudiging van FOBDTs, onafhankelijk van de volgorde van de kernen . . . . .	68
4.4	voegTypevoorwaardenToe( $S, x$ ): Het toevoegen van voorwaarden die volgen uit het type van een variabele. . . . .	70
4.5	BreidUitEnVoegVoorwToe( $S, \varphi$ ): Breidt Omega+ verzameling $S$ uit met voorwaarde $\varphi$ . . . . .	71
4.6	simpFOBDD: aritmetische vereenvoudiging van FOBDDs . . . . .	72

# Inleiding

Toen men lang geleden de eerste computer op de markt bracht, was de achterliggende motivatie dat dit apparaat kon helpen met het oplossen van allerlei problemen. Dit waren in het begin vaak complexe berekeningen, die voor mensen te tijdsconsumerend waren om zelf uit te voeren. Met een computer moesten mensen hun berekening niet meer zelf doen. Ze konden hun gegevens eraan meegeven, ertegen zeggen wat er met die gegevens moest gebeuren en de computer het werk laten doen. De mogelijkheden van dat “apparaat” zijn op relatief korte tijd zo enorm toegenomen, dat we nu voor bijna elk probleem dat we willen oplossen een computer gebruiken. Computers zijn zelfs al zo ver geëvolueerd dat er al problemen bestaan waarvoor we niet meer moeten zeggen tegen een computer hoe het moet worden opgelost, enkel hoe het probleem eruit ziet. Dit is natuurlijk een heel interessant gegeven, aangezien er veel alledaagse problemen zijn, die we gemakkelijk kunnen specificeren, maar waar we geen goede oplossingsstrategie voor kennen. Neem bijvoorbeeld allerlei planningsproblemen, zoals het plannen van een uurrooster aan een universiteit, waarbij rekening moet gehouden worden met beschikbaarheid van docenten, studenten, lokalen, . . . en allerlei andere voorwaarden. Als het gaat over 5 klassen, 3 leerkrachten en 4 lokalen, kunnen mensen deze planning zonder problemen zelf uitvoeren. In de praktijk zijn dit soort problemen echter vaak zo groot dat er geen overzicht te behouden valt. Als we nu een systeem zouden hebben waaraan we al onze kennis hierover zouden kunnen meegeven, waarmee dat systeem dan zou redeneren tot het een aanvaardbare oplossing had, zou dat het plannen van die uurroosters veel draaglijker maken.

Dit soort systemen bestaan gelukkig al, ze worden Knowledge-Based Systems (KBS) genoemd en er wordt heel veel onderzoek naar gedaan. Dit onderzoek spitst zich toe op twee aspecten. Enerzijds willen we dat een KBS zo efficiënt en snel mogelijk kan werken. Er wordt gezocht naar goede methoden om oplossingsstrategieën te bepalen bij willekeurige problemen. Hoe beter deze strategieën zijn, hoe sneller we problemen kunnen oplossen en hoe groter de problemen worden die we kunnen oplossen. Anderzijds wordt er ook onderzoek gedaan naar de manier waarop kennis kan worden overgebracht naar het systeem. We willen kennis kunnen weergeven op een manier die voor zowel mensen als de computer duidelijk is, zodat de computer er snel mee aan de slag kan.

Eén van de plaatsen waar dit onderzoek gedaan wordt, is in de onderzoeksgroep KRR van het departement computerwetenschappen aan de KU Leuven. Er wordt een KBS ontwikkeld met de naam IDP. In IDP kunnen problemen en kennis gespecificeerd worden met een uitbreiding van eerste-ordeloga,  $FO(.)$  genoemd. Deze taal is enerzijds herkenbaar en heeft een erg duidelijke informele semantiek, net zoals eerste-ordeloga, maar anderzijds



is deze taal toch erg expressief. We kunnen er namelijk concepten in uitdrukken die we niet in eerste-orde logica kunnen uitdrukken (zoals we later nog zullen zien). Eén van de specifieke types problemen die we met IDP kunnen oplossen is een modelexpansie. In een modelexpansie willen we een oplossing vinden voor een probleem dat bestaat uit een aantal voorwaarden, waar we al een gedeeltelijke oplossing voor hebben. Een sudoku is een goed voorbeeld van een dergelijk probleem. Een sudoku bestaat uit 81 vakjes, die zijn opgedeeld in rijen, kolommen en blokken. We hebben daarbij ook 9 getallen, die we in die vakjes moeten invullen volgens een aantal regels. Hierbij krijgen we voor sommige vakjes al gegeven welk nummer er moet instaan. We zoeken dan een invulling voor alle andere vakjes, zodat we een volledige reglementair ingevulde sudoku krijgen.

In deze thesis wordt het uitvoeren van zo een modelexpansie eens van dichterbij bekeken (hoofdstuk 3.1). Het algoritme dat in IDP gebruikt wordt, wordt besproken en uitgewerkt. Een deel van dit algoritme bestaat erin een eerste-orde theorie om te zetten naar een equivalente propositionele theorie. Dit wordt *grounden* genoemd. We kijken specifiek hoe we een *grounding* kunnen uitvoeren (hoofdstuk 3.1.3) en we onderzoeken enkele manieren om dit zo efficiënt mogelijk te doen. In IDP wordt hiertoe onder meer een speciale normaalvorm van formules gebruikt, die bijdragen aan de efficiëntie van het systeem. Deze formules worden FOBDDs (Binary Decision Diagrams) genoemd, hebben een binaire boomstructuur en worden geïntroduceerd in hoofdstuk 2.

Deze FOBDDs zijn een heel interessante normaalvorm voor formules in eerste-orde logica, onder meer omdat vereenvoudigingen er heel gemakkelijk in door te voeren zijn en omdat ze de eigenschap hebben dat twee FOBDDs voor twee equivalente eerste-orde formules vaak syntactisch gelijk zijn. Dit resultaat kan natuurlijk niet voor elke paar eerste-orde formules gelden, want dat zou betekenen dat we een manier gevonden hebben om satisfieerbaarheid van een willekeurige formule in eerste-orde logica te beslissen. We kunnen namelijk gewoon die formule omzetten naar een FOBDD en kijken of het de FOBDD voor false is. Eén van de dingen die er bijvoorbeeld voor zorgen dat twee equivalente FOBDDs toch verschillend zijn, is het bevatten van redundante deelFOBDDs.

In de context van deze thesis werd getracht de hoeveelheid redundantie in deze FOBDDs te verminderen. Dit wordt uitgewerkt specifiek in de context van het gebruik van aritmetische formules. Er werd gezocht naar een manier om te kunnen beslissen of een bepaalde aritmetische formule al dan niet satisfieerbaar is. Aangezien dit voor willekeurige aritmetische formules onbeslisbaar is, werd een specifieke deelklasse van de aritmetische formules bekeken, namelijk de aritmetische formules zonder vermenigvuldiging. De theorie van deze klasse wordt Presburger aritmetiek genoemd en is beslisbaar. Dit wordt bewezen in hoofdstuk 4.3.1. Met deze beslisbare theorie wordt vervolgens een algoritme ontwikkeld in hoofdstuk 4. Ter afsluiting van deze thesis wordt dit algoritme kort in de praktijk getest en worden er nog wat mogelijke uitbreidingen bekeken.

# Hoofdstuk 1

## Voorbeschouwing

### 1.1 Elementaire begrippen

Om overzicht te behouden in de rest van deze thesis, worden hier verschillende elementaire notaties en concepten uitgewerkt, die we verder in de thesis nodig hebben. De belangrijkste definities die we hier introduceren zijn de logica's: propositielogica en eerste-orde logica. In het algemeen zien we een logica als een systeem om te kunnen redeneren. Dit wil dus zeggen dat een logica een taal is, en dus een vocabularium (de symbolen die we mogen gebruiken), en een syntax (hoe we zinnen in de logica creëren uit deze symbolen) bezit. Om te kunnen redeneren met deze zinnen, hebben we ten slotte ook een aantal regels nodig, die ons kunnen vertellen of een gegeven zin al dan niet waar is. Deze verzameling regels noemen we de (formele) semantiek van de logica.

Interessant aan propositie en eerste-orde logica, is dat deze formele semantiek erg goed overeenstemt met de informele semantiek. Met andere woorden, de regels die we hieronder gaan definiëren, die zeggen of een formule al dan niet waar is, komen goed overeen met wat we intuïtief zouden verwachten. Een formule van de vorm  $\forall x(P(x) \vee Q(x))$ , wilt bijvoorbeeld zeggen dat gegeven een willekeurige waarde voor  $x$  ( $\forall x$ ) ofwel  $P(x)$  ofwel  $Q(x)$  waar is, dus exact zoals we het zouden lezen.

#### 1.1.1 Propositielogica

Om een logica te definiëren hebben we dus een vocabularium  $\Sigma$ , een syntax en semantiek nodig. We definiëren een vocabularium  $\Sigma = \{p_0, p_1, \dots\}$  als een verzameling propositie-symbolen (soms noemen we deze symbolen ook wel (propositie-)variabelen). De syntax vertelt ons dan welke formules behoren tot een logica. We definiëren deze inductief met behulp van een BNF-vorm ([ASU86]):

**Definitie 1.1.** Een formule in propositiologica over een vocabularium  $\Sigma$ , is van de vorm:

$F, G :=$	<b>f</b>	(false)
	<b>t</b>	(true)
	$a$	(voor alle $a \in \Sigma$ )
	$\neg F$	(negatie)
	$(F \wedge G)$	(conjunctie)
	$(F \vee G)$	(disjunctie)

De verzameling van al deze formules over een vocabularium  $\Sigma$  noteren we met  $\mathcal{L}_{\mathbf{Pr}, \Sigma}$ . Soms worden er ook nog de connectoren  $F \Rightarrow G$  (implicatie), en  $F \Leftrightarrow G$  (equivalentie) gebruikt, maar dit zijn afkortingen voor  $G \vee \neg F$  respectievelijk  $(F \wedge G) \vee (\neg F \wedge \neg G)$ .

Nu rest ons enkel nog de semantiek van de propositiologica te definiëren. Hiervoor hebben we eerst het concept van een valuatie nodig.

**Definitie 1.2.** Gegeven een vocabularium  $\Sigma = \{p_0, p_1, \dots\}$ , dan noemen we elke functie  $V : \Sigma \rightarrow \{\mathbf{t}, \mathbf{f}\}$  een  $\Sigma$ -valuatie. We gebruiken de notatie  $V = \{q_0, q_1, \dots, q_m\}$ , waarbij  $\{q_0, q_1, \dots, q_m\} \subset \Sigma$  en waarmee we bedoelen dat  $V(p) = \mathbf{t}$  voor alle  $p \in \{q_0, q_1, \dots, q_m\}$  en  $V(p) = \mathbf{f}$  voor alle  $p \in \Sigma \setminus \{q_0, q_1, \dots, q_m\}$ .

Gegeven een valuatie  $V$ , kunnen we de semantiek voor de propositiologica definiëren door de relatie  $\models$ . We schrijven  $V \models \varphi$ , waarmee we bedoelen dat de formule  $\varphi$  geldt, gegeven een valuatie  $V$ . We definiëren deze relatie recursief (stel  $p \in \Sigma$ , en  $\varphi, \varphi_1, \varphi_2$  propositionele formules).

$V \models p$	als $V(p) = \mathbf{t}$
$V \models \neg \varphi$	als $V \not\models \varphi$
$V \models \varphi_1 \wedge \varphi_2$	als $V \models \varphi_1$ en $V \models \varphi_2$
$V \models \varphi_1 \vee \varphi_2$	als $V \models \varphi_1$ of $V \models \varphi_2$

We zeggen nu dat een propositionele formule  $\varphi$  waar is in  $V$ , als  $V \models \varphi$ . We zeggen dan omgekeerd ook dat  $V$  een model is voor  $\varphi$ . Bij uitbreiding zeggen we dat een theorie  $\mathcal{T}$  (eindige verzameling formules) waar is in  $V$  als elke formule in  $\mathcal{T}$  waar is in  $V$ . We noemen  $V$  dan een  $\mathcal{T}$ -model. Merk op dat het waar zijn van een theorie neerkomt op het waar zijn van de conjuncties van alle formules in die theorie.

Een veelgebruikt concept voor theorieën in propositiologica is satisfieerbaarheid.

**Definitie 1.3.** We noemen een propositionele formule  $\varphi$  **satisfieerbaar**, als er een valuatie  $V$  bestaat zodat  $\varphi$  waar is in  $V$ . We noemen  $\varphi$  een **tautologie** als  $\varphi$  waar is voor elke structuur  $V$ . We noemen een propositionele theorie **satisfieerbaar** als er een  $\mathcal{T}$ -model  $V$  bestaat.

Merk op dat hieruit volgt dat een formule  $\varphi$  satisfieerbaar is als en slechts als  $\neg \varphi$  geen tautologie is. We verduidelijken het concept van satisfieerbaarheid met volgend voorbeeld:

**Voorbeeld 1.1.** Volgende theorie is een voorbeeld van een satisfieerbare theorie:

$$\begin{aligned} p_0 \wedge p_1 \\ \neg p_0 \vee \neg p_2 \end{aligned}$$

Er bestaat namelijk een model voor deze theorie, namelijk  $V = \{p_0, p_1\}$ . Volgende theorie is dan een voorbeeld van een niet-satisfieerbare theorie

$$\begin{aligned} p_0 \wedge p_1 \\ \neg p_0 \wedge \neg p_2 \end{aligned}$$

Er bestaat geen enkel model  $V$  zodat  $V \models \mathcal{T}$ .

Voor een propositionele formule  $\varphi$  definiëren we ook nog drie veelgebruikte normaalvormen<sup>1</sup>:

**Definitie 1.4.** Een propositionele formule  $\varphi$  is in **Negatie Normaalvorm (NNF)** als elk negatie symbool in  $\varphi$  enkel voorkomt vlak voor een propositiesymbool.

**Definitie 1.5.** Een propositionele formule  $\varphi$  is in **Conjunctieve Normaalvorm (CNF)** als  $\varphi$  geschreven is als een conjunctie van disjuncties van (negaties van) propositiesymbolen, dus  $\varphi$  is van de vorm:

$$\varphi = \bigwedge_i \bigvee_j p_{i,j}.$$

Waarbij  $p_{i,j}$  een propositiesymbool is, of de negatie ervan.

**Definitie 1.6.** Een propositionele formule  $\varphi$  is in **Disjunctieve Normaalvorm (DNF)** als  $\varphi$  geschreven is als een disjunctie van conjuncties van (negaties van) propositiesymbolen, dus  $\varphi$  is van de vorm:

$$\varphi = \bigvee_i \bigwedge_j p_{i,j}.$$

Waarbij  $p_{i,j}$  een propositiesymbool is, of de negatie ervan.

Het gebeurt regelmatig dat we een gegeven formule willen omzetten naar een formule in zijn conjunctieve normaalvorm. De meest voor de hand liggende methode is om dit te doen door de distributiviteitseigenschap van de eerste-orde logica toe te passen, maar dit kan een formule exponentieel in lengte doen toenemen ([MVFH03]). In sommige gevallen, wanneer we niet echt een equivalente theorie zoeken, zijn er betere technieken, zoals de Tseitin transformatie ([Tse68]). Tseitin bewees ook dat de Tseitin transformatie van een gegeven theorie  $\mathcal{T}$  equi-satisfiebaar is aan de theorie zelf (propositie 1.1).

**Propositie 1.1.** Noem  $\mathcal{T}$  een willekeurige theorie, en  $\mathcal{T}'$  de Tseitin transformatie van  $\mathcal{T}$ , volgens algoritme 1.1. Dan geldt dat elk model van  $\mathcal{T}'$  ook een model is voor  $\mathcal{T}$  en dat elk model voor  $\mathcal{T}$  uitgebreid kan worden tot een model voor  $\mathcal{T}'$ . We noemen  $\mathcal{T}$  en  $\mathcal{T}'$  **equi-satisfiebaar**, want  $\mathcal{T}$  is satisfiebaar als en slechts als  $\mathcal{T}'$  satisfiebaar is.

<sup>1</sup>Merk op dat formules in CNF of DNF automatisch ook in NNF staan, of met andere woorden: CNF en DNF zijn sterkere normaalvormen dan NNF

---

**Algoritme 1.1:** Tseitin Transformatie

---

**input** : Een propositionele theorie  $\mathcal{T}$ **output**: Een propositionele theorie in CNF-normaal vorm  $\mathcal{T}'$ 

```

1 for  $\varphi \in \mathcal{T}$  do
2   | Propageer negaties tot vlak voor atomen in  $\varphi$ ;
3 end
4 for  $\varphi \in \mathcal{T}$  do
5   | if  $\varphi = \varphi_1 \wedge \varphi_2 \wedge \dots \wedge \varphi_n$  then
6     |  $\mathcal{T}.verwijder(\varphi)$ ;
7     | for  $i = 1..n$  do
8       |  $\mathcal{T}.voegToe(\varphi)$ ;
9     | end
10  |
11 end
12 while  $\mathcal{T}$  niet in CNF do
13   | for  $\varphi \in \mathcal{T}$  do
14     | if ( $\varphi$  is van de vorm  $\varphi_1 \vee \varphi_2 \vee \dots \vee \varphi_n$ )
15     | en  $\exists \varphi_i$  van de vorm  $\psi_1 \wedge \dots \wedge \psi_m$  then
16       |  $\mathcal{T}.verwijder(\varphi)$ ;
17       |  $P$ =nieuwe variabele;
18       |  $\chi = \varphi_1 \vee \varphi_2 \vee \dots \vee \varphi_{i-1} \vee P \vee \varphi_{i+1} \vee \dots \vee \varphi_n$ ;
19       |  $\mathcal{T}.voegToe(\chi)$ ;
20       |  $\varepsilon = (\neg\psi_1 \vee \dots \vee \neg\psi_m \vee P)$ ;
21       | Propageer negaties tot vlak voor atomen in  $\varepsilon$ ;
22       |  $\mathcal{T}.voegToe(\varepsilon)$ ;
23       | for  $j = 1..n$  en  $j \neq i$  do
24         |  $\omega = \neg P \vee \psi_j$ ;
25         | Propageer negaties tot vlak voor atomen in  $\omega$ ;
26         |  $\mathcal{T}.voegToe(\omega)$ ;
27       | end
28     |
29   | end
30 end
31 return  $\mathcal{T}$ ;

```

---

Merk op dat dit niet hetzelfde is als equivalentie. Bekijk bijvoorbeeld volgende formule  $\mathcal{T} = \{p_0 \vee (p_1 \wedge p_2)\}$  met Tseitin transformatie  $\mathcal{T}' = \{p_0 \vee P, \neg P \vee p_1, \neg P \vee p_2, \neg B \vee \neg C \vee P\}$ . Dan is  $V = \{p_0, P\}$  een model voor  $\mathcal{T}$  over  $\Sigma = \{p_0, p_1, p_2, P\}$ , maar geen model voor  $\mathcal{T}'$ , aangezien bijvoorbeeld  $\neg P \vee p_1$  niet geldt.

### 1.1.2 Eerste-ordeloga

Zoals bij het definiëren van de propositiologica, moeten we opnieuw een vocabularium, een syntax en een semantiek definiëren voor de eerste-ordeloga.

Het vocabularium  $\Sigma$  bestaat uit 3 verzamelingen: een verzameling

$$\Sigma_{pred} = \{P_1/n_1, P_2/n_2, \dots\}$$

van predicaten (waarbij  $n_i$  de ariteit van  $P_i$  voorstelt), een verzameling

$$\Sigma_{func} = \{F_1/m_1, F_2/m_2, \dots\}$$

van functies (waarbij  $m_i$  de ariteit van  $F_i$  voorstelt)<sup>2</sup> en een verzameling

$$\Sigma_{var} = \{x, y, \dots\}$$

van variabelen. Als we voor een specifiek voorbeeld het vocabularium willen geven, zullen we de variabelen meestal weglaten, omdat dit duidelijk zal zijn uit de context van de semantiek.

De syntax wordt dan als volgt gedefinieerd:

**Definitie 1.7.** Een term in eerste-ordeloga over een vocabularium  $\Sigma$ , is van de vorm:

$$\begin{aligned} t &:= x && \text{(een variabele is een term)} \\ &F(t_1, t_2, \dots, t_n) && \text{(Functie toepassen)} \end{aligned}$$

Met alle  $t_i$  termen, en  $F/n \in \Sigma_{func}$ .

**Definitie 1.8.** Een formule in eerste-ordeloga over een vocabularium  $\Sigma$ , is van de vorm:

$$\varphi, \psi := \neg \varphi \quad \text{(negatie)} \quad (1.1)$$

$$(\varphi \wedge \psi) \quad \text{(conjunctie)} \quad (1.2)$$

$$(\varphi \vee \psi) \quad \text{(disjunctie)} \quad (1.3)$$

$$\forall x(\varphi) \quad \text{(Universele kwantificatie)} \quad (1.4)$$

$$\exists x(\varphi) \quad \text{(Universele kwantificatie)} \quad (1.5)$$

$$P(t_1, t_2, \dots, t_n) \quad \text{(Predicaat Toepassen)} \quad (1.6)$$

$$t_1 = t_2 \quad \text{(Gelijkheidsrelatie)} \quad (1.7)$$

Met alle  $t_i$  termen.

We noemen verder een formule van type (1.6) of (1.7) ook wel een **atomische formule**. De verzameling van alle formules in de eerste-ordeloga over een vocabularium  $\Sigma$  noteren we met  $\mathcal{L}_{\mathbf{FO}, \Sigma}$

---

<sup>2</sup>Merk op dat we ook constanten kunnen gebruiken in eerste orde logica, dit zijn gewoon functies met ariteit 0.

**Definitie 1.9.** Een formule waar geen vrije variabelen in voor komen (dus geen variabelen die niet gebonden zijn door een kwantor), noemen we een **gesloten formule** of een **zin**. Een verzameling zinnen, noemen we een **theorie**.

Bij het definiëren van de semantiek voor propositiologica, moesten we eerst een valuatie definiëren: een interpretatie voor de propositiesymbolen. Voor eerste-ordeloga moeten we iets gelijkaardigs doen. We definiëren eerst een structuur: een interpretatie voor de predicaten en de functies.

**Definitie 1.10.** Een  $\Sigma$ -structuur  $I$  bestaat uit een domein  $D$ , waarbij  $D$  een verzameling elementen is waar we onze variabelen door kunnen substitueren<sup>3</sup> en een afbeelding die:

- elke  $P/n \in \Sigma_{pred}$  op een relatie  $R_P \subset D^n$  afbeeldt. We noemen  $R_P$  de interpretatie van  $P$ .
- elke  $F/m \in \Sigma_{func}$  op een functie  $G_F \subset D^m \rightarrow D$  afbeeldt. We noemen  $G_F$  de interpretatie van  $F$ .

We noemen een  $\Sigma$ -structuur  $I$  eindig als  $D$  eindig is.

We hebben enkel nog een interpretatie voor de variabelen nodig, voor we de semantiek kunnen definiëren.

**Definitie 1.11.** Een variabele-interpretatie  $\omega$  voor  $I$  ( $\Sigma$ -structuur) is een afbeelding die elke  $x \in \Sigma_{var}$  op een  $d \in D$  afbeeldt. Aangezien het voor elk element in het vocabularium duidelijk is, of we het moeten interpreteren met  $I$ , of met  $\omega$  noteren we  $I\omega$  voor de afbeelding vanuit  $\Sigma$  die elk element interpreteert. We noemen deze afbeelding dan ook kortweg de interpretatie.

We gebruiken nu deze structuur opnieuw om een relatie te construeren voor alle termen en formules, zoals we bij propositiologica gedaan hebben. Eerst doen we dit voor termen (met  $t_1, \dots, t_n$  bedoelen we termen, en met  $x$  een variabele in  $\Sigma$ ):

$$\begin{aligned} I\omega(x) &= \omega(x) \\ I\omega(F(t_1, \dots, t_n)) &= G_F(I\omega(t_1), \dots, I\omega(t_n)) \end{aligned}$$

Voor formules definiëren we de relatie  $I\omega \models \varphi$  die we kunnen lezen als “ $\varphi$  is waar in  $I$ , als we de variabelen interpreteren volgens  $\omega$ ”. We doen dit opnieuw recursief (we bedoelen

---

<sup>3</sup>Gegeven een formule  $\varphi[x]$  met vrije variabele  $x$ , en een domein  $D = \{d_0, \dots, d_n\}$  (domeinelementen), dan noteren we  $\varphi[x]$  met  $x$  vervangen door  $d_i$  als  $\varphi[x/d_i]$ .

met  $t_1, \dots, t_n$  termen en met  $\varphi, \varphi_1, \varphi_2, \dots$  formules).

$I\omega \models P(t_1, \dots, t_n)$	als $(I\omega(t_1), I\omega(t_2), \dots, I\omega(t_n)) \in R_P$
$\models t_1 = t_2$	als $I\omega(t_1) = I\omega(t_2)$
$\models \neg\varphi$	als $I\omega \not\models \varphi$
$\models \varphi_1 \wedge \varphi_2$	als $I\omega \models \varphi_1$ en $I\omega \models \varphi_2$
$\models \varphi_1 \vee \varphi_2$	als $I\omega \models \varphi_1$ of $I\omega \models \varphi_2$
$\models \forall x(\varphi)$	als $I\omega[x/d] \models \varphi$ voor alle $d \in D$
$\models \exists x(\varphi)$	als er een $d \in D$ bestaat zodat $I\omega[x/d] \models \varphi$ ,

We noemen  $I\omega$  een model voor  $\varphi$  als  $I\omega \models \varphi$ . Merk op dat voor een formule  $\varphi$  zonder vrije variabelen, geldt dat het waar zijn in een interpretatie, onafhankelijk is van de variabele-interpretatie. We zeggen dan ook dat  $\varphi$  waar is in  $I$ , dus  $I \models \varphi$ , en bij uitbreiding dat  $I$  een model is voor een theorie  $\mathcal{T}$  als elke zin in  $\mathcal{T}$  waar is in  $I$ . We noteren dit als  $I \models \mathcal{T}$ . We misbruiken deze notatie soms voor twee theorieën  $\mathcal{T}_1$  en  $\mathcal{T}_2$ , door  $\mathcal{T}_1 \models \mathcal{T}_2$  te schrijven, waarmee we bedoelen dat elk model voor  $\mathcal{T}_1$  ook een model is voor  $\mathcal{T}_2$ .

Net zoals bij propositiologica bestaan er voor formules in eerste-ordeloga verschillende normaalvormen, we definiëren drie vaak gebruikte vormen<sup>4</sup>:

**Definitie 1.12.** Een eerste-ordeformule  $\varphi$  is in **Negatie Normaalvorm (NNF)** als elk negatie symbool in  $\varphi$  enkel voorkomt vlak voor een atomische formule.

**Definitie 1.13.** Een eerste-ordeformule  $\varphi$  is in **Conjunctieve Normaalvorm (CNF)** als  $\varphi$  geschreven is als

$$Q_1x_1, Q_2x_2, \dots, Q_nx_n\psi'[x_1, \dots, x_n],$$

met  $Q_1, Q_2, \dots, Q_n$  kwantoren,  $\psi'[x_1, \dots, x_n]$  kwantorvrij, en  $\psi'$  een conjunctie van disjuncties van atomische formules is.

**Definitie 1.14.** Een eerste-ordeformule  $\varphi$  is in **Disjunctieve Normaalvorm (DNF)** als  $\varphi$  geschreven is als

$$Q_1x_1, Q_2x_2, \dots, Q_nx_n\psi'[x_1, \dots, x_n],$$

met  $Q_1, Q_2, \dots, Q_n$  kwantoren,  $\psi'[x_1, \dots, x_n]$  kwantorvrij, en  $\psi'$  een disjunctie van conjuncties van atomische formules is.

We hebben voor eerste-ordeloga ook een tegenhanger van definitie 1.3:

**Definitie 1.15.** We noemen een eerste-ordeformule  $\varphi$  **satisfieerbaar**, als er een structuur  $I$  en een variabele-interpretatie  $\omega$  bestaat zodat  $\varphi$  waar is in  $I$ , met de variabelen geïnterpreteerd door  $\omega$ . We noemen  $\varphi$  een **tautologie** als  $\varphi$  waar is voor elke structuur  $I$  en elke variabele-interpretatie  $\omega$ . Een eerste-ordetheorie  $\mathcal{T}$  noemen we **satisfieerbaar** als er een  $\mathcal{T}$ -model  $V$  bestaat.

<sup>4</sup>Ook voor eerste-ordeloga geldt dat CNF en DNF sterkere normaalvormen zijn dan NNF.



### 1.1.3 Uitbreidingen op eerste-ordeloga

In dit hoofdstuk bekijken we ten slotte nog enkele uitbreidingen op eerste-ordeloga. Sommige uitbreidingen zijn eerder praktisch van aard, zoals bijvoorbeeld het invoeren van aritmetiek, en brengen niets bij aan de expressiviteit van eerste-ordeloga. Andere uitbreidingen zijn fundamenteeler, en voegen essentieel iets toe aan eerste-ordeloga, zoals bijvoorbeeld inductieve definities.

**Aritmetiek** Het toevoegen van aritmetiek is een uitbreiding van het eerste type. Deze uitbreiding brengt niets bij aan de expressiviteit van de logica, maar het is daarom niet minder zinvol om de uitbreiding toe te voegen. In het begin van hoofdstuk 1.1.2 hebben we vermeld dat het ook belangrijk is voor een logica om leesbaar te zijn, dat er een duidelijk verband is tussen de formele en de informele semantiek. Als we nu iets willen uitdrukken over aritmetiek, dan willen we bijvoorbeeld niet telkens opnieuw een functie moeten definiëren die twee variabelen afbeeldt op hun som. We willen dat dit ingebouwd zit in de logica, zodat we dit altijd onmiddellijk kunnen gebruiken. We gaan hier niet dieper in op de precieze inbedding van aritmetiek in eerste-ordeloga. In de inleiding van hoofdstuk 4.3.1 doen we dit wel, en bewijzen we ook enkele eigenschappen die ons vertellen dat de functies en predicaten die we introduceren om de aritmetiek weer te geven zich ook gedragen zoals we gewoon zijn.

**Getypeerde eerste-ordeloga** We kunnen voor predicaten, functies en variabelen in FO, types definiëren. Daarmee kunnen we het domein van argumenten vastleggen op een deelverzameling van het volledige probleemdomein. We definiëren deze types door een nieuwe verzameling  $\Sigma_{type} = \{s_1, s_2, \dots\}$  van types aan ons vocabularium toe te voegen. We definiëren dan predicaten als

$$\Sigma_{pred} = \{P_1(s_{1,1}, s_{1,2}, \dots), P_2(s_{2,1}, s_{2,2}, \dots), \dots\}$$

en functies als

$$\Sigma_{func} = \{F_1(s_{1,1}, s_{1,2}, \dots) : s_1, F_2(s_{2,1}, s_{2,2}, \dots) : s_2, \dots\}$$

Hierbij geeft elke  $s_{i,j} \in \Sigma_{type}$ , het verwachte type aan, en  $s_i \in \Sigma_{type}$  geeft het type aan dat de functie teruggeeft<sup>5</sup>. Een structuur en variabele-interpretatie gaan we analoog herdefiniëren, door bij de structuur elk type  $s$  op een niet-lege verzameling  $D_s$  af te beelden. We noemen dan  $D_s$  het domein van  $s$ . De interpretatie van een predicaat  $P(s_1, \dots, s_n)$  is dan niet langer een deelverzameling van  $D^n$ , maar van  $D_{s_1} \times D_{s_2} \times \dots \times D_{s_n}$ . Analoog passen we de interpretaties voor variabelen en functies aan. We noteren deze uitbreiding van FO als FO[Type]<sup>6</sup>.

<sup>5</sup>Alle variabelen hebben ook een eigen type, maar om notationale redenen schrijven we er dit niet altijd bij. We spreken af dat we dit kunnen opvragen met een afbeelding  $type(x)$ .

<sup>6</sup>We gaan vanaf nu altijd deze variant van FO gebruiken (met de aanpassing van de subtypes). Dus verder wordt met FO altijd FO[Type] bedoeld. De aangepaste definities van FO[Type] en zijn structuur en interpretaties worden uitgewerkt in bijlage A

**Geordende Getypeerde eerste-ordeloga** De uitbreiding van hierboven kunnen we nog verder uitbreiden, namelijk met het idee van subtypes. Stel dat we de een getallensysteem willen modelleren, dan hebben we verschillende types nodig, zoals  $\mathbb{R}$ ,  $\mathbb{Q}$ ,  $\mathbb{Z}$  en  $\mathbb{N}$ . Met de hierboven voorgestelde uitbreiding kunnen we niet modelleren dat een natuurlijk getal ook een geheel, rationaal en reëel getal is, aangezien een type uniek is. Door het introduceren van subtypes kunnen we deze opmerking al deels tegemoetkomen. We definiëren types zoals hierboven, maar met de opmerking dat types hiërarchisch gestructureerd zijn. Een rationaal getal kunnen we dan bijvoorbeeld automatisch ook bekijken als een reëel getal.

**Aggregaten** Aggregaten zijn functies op hele verzamelingen, zoals bijvoorbeeld het berekenen van de kardinaliteit, of het maximum van een verzameling getallen. We noteren FO met aggregaten als FO[Agg].

**Inductieve definities** Inductieve definities maken het mogelijk om een interpretatie van een element volledig te specificeren. Gewone eerste-ordeloga veronderstelt dat een element wat niet expliciet gespecificeerd wordt, zowel waar als onwaar kan zijn. Een inductieve definitie daarentegen, kunnen we wel gebruiken om de interpretatie voor dat element volledig te specificeren. Alles wat kan worden afgeleid uit de definitie is waar, en al de rest is onwaar. Eerste-ordeloga is normaal niet in staat om bijvoorbeeld concepten zoals de eindigheid van een universum, of eindigheid van een graaf uit te drukken, als gevolg van de compactheidsstelling ([End72])<sup>7</sup>. We noteren FO uitgebreid met inductieve definities als FO[ID] ([DT04]).

FO(.) Dit is niet zozeer een nieuwe uitbreiding, maar een verzamelnaam voor alle uitbreidingen die we zojuist gedefinieerd hebben, ingevoerd door [Wit10]. De  $\cdot$  in FO(.), staat voor de veralgemening van [Agg], [ID], [Type], ... uit de vorige uitbreidingen.

## 1.2 Kennisrepresentatie

Al deze logicas en uitbreidingen daarop zijn dan wel erg mooie wiskundige structuren, maar ze zijn eigenlijk veel meer dan dat. Hoe abstract ze ook zijn (of lijken) op het eerste zicht, ze kennen wel degelijk een heel toepassingsgebied. Het vakgebied van de kennisrepresentatie houdt zich bezig met het voorstellen van kennis op een manier die voor zowel mens als computer verstaanbaar is. Deze kennis kan dan door mensen aan computers worden gegeven, en de rekenkracht van computers kan dan gebruikt worden om te redeneren met deze kennis. Voor kennisrepresentatie is dus een taal nodig om de kennis weer te geven, en een inferentiesysteem dat kan redeneren met deze taal. We herkennen natuurlijk meteen de eigenschappen van een logica in de vereisten voor kennisrepresentatie.

---

<sup>7</sup>Voor een bewijs van deze bewering, zie (onder andere) [Lib04].

Een eerste mogelijkheid van een taal die we kunnen gebruiken voor kennisrepresentatie zou de eerste-ordeloga kunnen zijn, met daarbij de hierboven beschreven semantiek. Dit zou inderdaad perfect kunnen werken, maar we hebben hierboven ook verschillende beperkingen ontdekt op het vlak van expressiviteit of van leesbaarheid. In de praktijk worden daartoe vaak allerlei uitbreidingen van eerste-ordeloga gebruikt. De uitbreiding die hierboven geconstrueerd werd ( $\text{FO}(\cdot)$ ), wordt bijvoorbeeld gebruikt in het IDP-systeem, ontwikkeld door de onderzoeksgroep KRR (Knowledge Representation and Reasoning) van het departement Computerwetenschappen aan de KU Leuven ([MWD06]). Het IDP-systeem wordt verder besproken in hoofdstuk 3.1.2.

Een eerste specifieke manier om te redeneren, in het vakgebied van de kennisrepresentatie, is **modelgeneratie**. Gegeven een vocabularium  $\Sigma$ , een domein  $D = D_1 \times \dots \times D_n$  en een theorie  $\mathcal{T}$ , zoeken we een model dat aan deze theorie voldoet, in het geval dat dit bestaat. Informeel wil dit dus zeggen, dat we een beschrijving  $D$  van een situatie  $\Sigma$  hebben (bijvoorbeeld een verzameling studenten, docenten, vakken en lokalen), en een aantal voorwaarden  $\mathcal{T}$  (er kunnen bijvoorbeeld geen twee lessen tegelijk in hetzelfde lokaal doorgaan), waarbij we dan een model zoeken waar aan alle voorwaarden voldaan wordt (een lessenrooster).

Meer formeel:

**Definitie 1.16.** *Gegeven een eerste-ordetheorie  $\mathcal{T}$  over een eindig vocabularium  $\Sigma$ . Het zoeken van een model  $M$  voor  $\mathcal{T}$  over  $\Sigma$  noemen we **modelgeneratie**.*

Een tweede gelijkaardige manier van redeneren, noemen we modelexpansie. Het is gelijkaardig aan modelgeneratie, maar we hebben al een deel van een model, wat we willen uitbreiden tot een volledig model voor de theorie (dus stel dat er al gegeven is dat op maandagvoormiddag Algebra I moet gegeven worden in lokaal 05.23). We kunnen dit ook formeel definiëren:

**Definitie 1.17.** *Gegeven een eerste-ordetheorie  $\mathcal{T}$  over een eindig vocabularium  $\Sigma$ . Zij dan  $\sigma \subset \Sigma$ , en  $I$  een interpretatie voor  $\sigma$ . Het zoeken van een model  $M$  voor  $\mathcal{T}$  over  $\Sigma$ , zodat de beperking van  $M$  tot  $\sigma$  gelijk is aan  $I$ , noemen we **modelexpansie**. We noemen  $I$  de **inputstructuur**.*

**Opmerking:** We definiëren de relatie  $\subset$  (deelvocabularium van) tussen twee vocabularia ( $\sigma \subset \Sigma$ ) van een eerste-ordetheorie iets anders dan men zou verwachten. We zeggen dat een vocabularium  $\sigma \subset \Sigma$  als  $\sigma_{var} \subset \Sigma_{var}$ ,  $\sigma_{pred} \subset \Sigma_{pred}$ ,  $\sigma_{func} \subset \Sigma_{func}$  en (!)  $\sigma_{type} \stackrel{!}{=} \Sigma_{type}$ . Dit doen we onder andere zodat bij modelexpansie voor elk type het domein volledig wordt bepaald door de inputstructuur. Een andere reden is dat we anders ook niet zomaar deelverzamelingen van bijvoorbeeld de predicaten mogen nemen, omdat er predicaten kunnen inzitten die types gebruiken die niet in  $\sigma$  zitten.

In hoofdstuk 3.1 gaan we dieper in op modelexpansie- en modelgeneratieproblemen, we bekijken daar ook de specifieke werking van een modelexpansiealgoritme.

### 1.2.1 Voorbeelden

Gedurende deze thesis zullen we enkele voorbeelden voor modelexpansie en -generatie, enkele keren hergebruiken. Deze voorbeelden worden hier gedefinieerd en uitgewerkt.

**Voorbeeld 1.2.** Een battleship puzzel bestaat uit een rooster met  $n \times n$  vakjes, en een aantal schepen die we op dit rooster willen plaatsen (vocabularium  $\Sigma$ ). Hierbij krijgen we als voorwaarden voor elke rij/kolom een nummer wat voor elke rij/kolom het precieze aantal scheepsstukken op deze kolom voorstelt. Daarnaast is er ook nog de regel dat 2 schepen elkaar niet horizontaal, verticaal of diagonaal mogen raken (theorie  $\mathcal{T}$ ). Er kan al een deel van de oplossing gegeven worden, in de zin van geplaatste scheepsstukken of getekende zeeën (interpretatie  $I$  voor deelvocabularium  $\sigma$ ). De puzzel wordt dan opgelost door voor elke vakje te beslissen of het vakje water bevat of een stuk schip, en dat tweede geval, welk stuk van welk schip (model  $\mathcal{M}$  voor  $\Sigma$ ). Een voorbeeld van een battleship puzzel zien we in figuur 1.1.

**Voorbeeld 1.3.** Een sudoku-puzzel is natuurlijk een veel bekender voorbeeld van een modelexpansie-probleem. Gegeven een rooster van  $9 \times 9$  vakjes, ingedeeld in  $3 \times 3$  blokken van elk  $3 \times 3$  vakjes, is het de bedoeling om in elk vakje exact een nummer van 1 tot 9 in te vullen. Hierbij moet aan de volgende regels voldaan worden:

- Elk blok in het rooster moet elk nummer exact één keer bevatten
- Elke rij in het rooster moet elk nummer exact één keer bevatten
- Elke kolom in het rooster moet elk nummer exact één keer bevatten

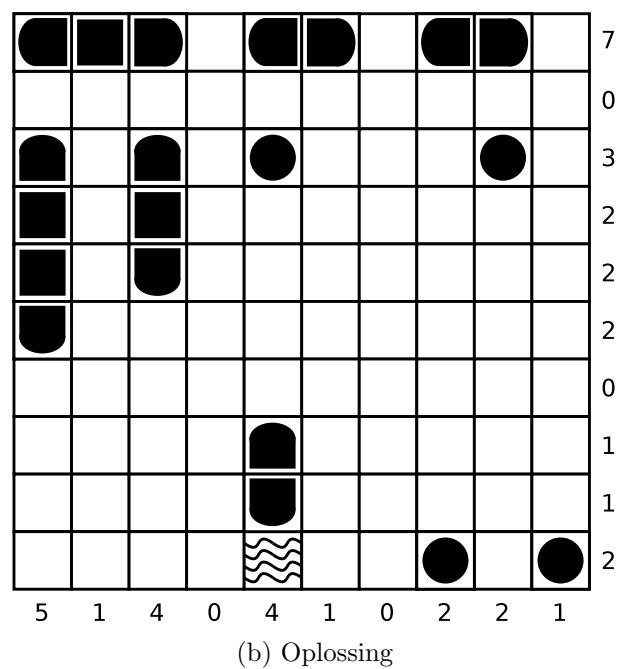
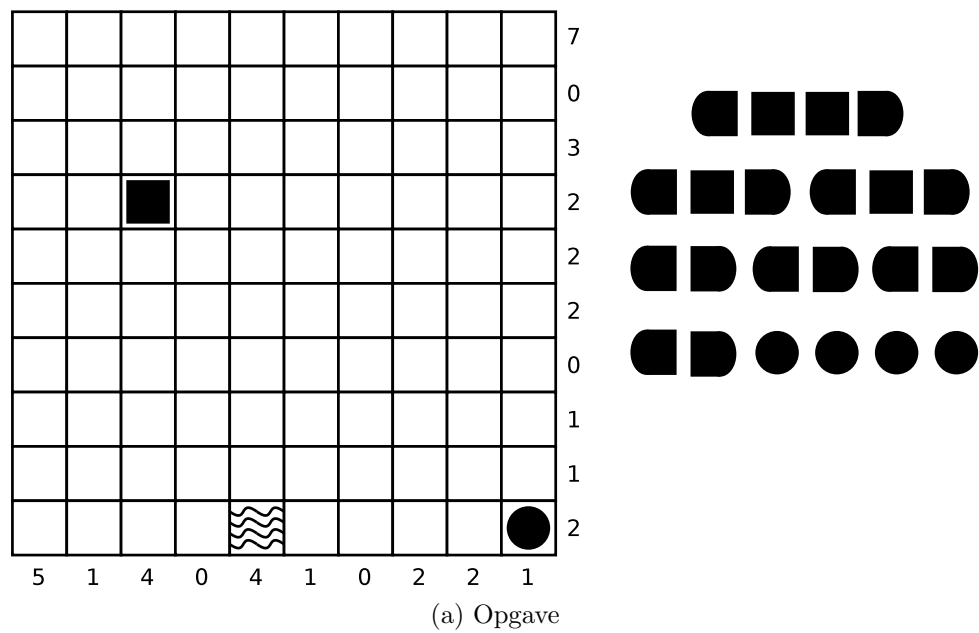
Er worden ook altijd al een aantal vakjes ingevuld, waarmee de puzzel verder opgebouwd kan worden. In figuur 1.2 zien we een grafische voorstelling van een sudoku.

Informeel is dit probleem in enkele zinnen gespecificeerd, maar als we het willen laten oplossen door een computer, moeten we het probleem ook formeel specificeren. Bij wijze van voorbeeld zullen we dit doen voor de sudoku-puzzel. We gebruiken hiervoor de uitgebreide versie van eerste-ordeloga (FO(.)). We moeten dus een vocabularium, deelvocabularium, theorie en inputstructuur definiëren.

Het vocabularium voor een sudoku is

$$\begin{aligned} \Sigma = \{ & \Sigma_{type}, \Sigma_{pred}, \Sigma_{func}, \Sigma_{var} \} \\ & \{ \{ B, Row, Column, Number \}, \\ & \{ Given(Row, Column, Number), Solution(Row, Column, Number) \}, \\ & \{ Block(Row, Column) : B \}, \{ \} \} \end{aligned} \quad (1.8)$$

Hierin is  $Solution(Row, Column, Number)$  een predicaat dat zegt dat  $Number$  op rij  $Row$  en kolom  $Column$  staat. Het predicaat  $Given(Row, Column, Number)$  zegt eigenlijk hetzelfde, maar we gebruiken dit predicaat om de beginvoorwaarden te geven. De functie



Figuur 1.1: Het battleshipprobleem

$Block(Row, Column) : B$  beeldt een vakje af op het blok waar het zich in bevindt. Zij dan

$$\begin{aligned}\sigma = \{ & \{Row, Column, number\}, \\ & \{Given(Row, Column, Number)\}, \\ & \{Block(Row, Column) : B\}, \{\}\end{aligned}\quad (1.9)$$

een deelverzameling van  $\Sigma$ , en laat  $I$  een interpretatie zijn voor  $\sigma$ . Dit wil eigenlijk zeggen dat we een domein hebben voor  $B$ ,  $Row$ ,  $Column$  en  $Number$  (in een standaard sudoku is dit voor alle vier  $[1..9] \subset \mathbb{N}$ ). We hebben ook al een interpretatie voor  $Given(Row, Column, Number)$  en  $Block(Row, Column) : B$ . De theorie  $\mathcal{T}$  bestaat dan uit volgende zinnen:

$$\forall x, y, z (Given(x, y, z) \Rightarrow Solution(x, y, z)) \quad (1.10)$$

$$\forall x, y (\exists! z (Solution(x, y, z))) \quad (1.11)$$

$$\forall x, z (\exists! y (Solution(x, y, z))) \quad (1.12)$$

$$\forall y, z (\exists! x (Solution(x, y, z))) \quad (1.13)$$

$$\forall b, z (\exists! x, y (b = Block(x, y) \wedge Solution(x, y, z))) \quad (1.14)$$

Het model  $I$  voor ons deelvocabularium bestaat dus uit een interpretatie voor de types en het  $Given$ -predicaat.

$$\begin{aligned}I(B) &= \{1, 2, 3, 4, 5, 6, 7, 8, 9\} \\ I(Row) &= \{1, 2, 3, 4, 5, 6, 7, 8, 9\} \\ I(Column) &= \{1, 2, 3, 4, 5, 6, 7, 8, 9\} \\ I(Number) &= \{1, 2, 3, 4, 5, 6, 7, 8, 9\} \\ I(Given(Row, Column, Number)) &= \{(1, 1, 3), (1, 3, 9), (1, 4, 8), (1, 5, 7), (1, 9, 4), \\ & \quad (2, 6, 5), (2, 9, 8), \\ & \quad (3, 1, 8), (3, 2, 7), (3, 4, 4), \\ & \quad (4, 1, 1), (4, 3, 4), (4, 4, 5), (4, 5, 8), (4, 9, 3), \\ & \quad (5, 4, 7), (5, 6, 6), \\ & \quad (6, 1, 7), (6, 5, 3), (6, 6, 4), (6, 7, 1), (6, 9, 5), \\ & \quad (7, 6, 9), (7, 8, 8), (7, 9, 1), \\ & \quad (8, 1, 9), (8, 4, 3), \\ & \quad (9, 1, 4), (9, 5, 5), (9, 6, 7), (9, 8, 2), (9, 9, 6)\}\end{aligned}$$

Modelexpansie voor deze theorie, komt dus neer op het zoeken van een interpretatie  $M$  (invullen van nummers in de vakjes), dat voldoet aan de theorie  $\mathcal{T}$  (de regels van sudoku), en waarvoor  $M$  beperkt tot  $\sigma$  gelijk is aan  $I$  (voldoet aan de beginvoorwaarden).

<b>3</b>		<b>9</b>	<b>8</b>	<b>7</b>				<b>4</b>
					<b>5</b>			<b>8</b>
<b>8</b>	<b>7</b>		<b>4</b>					
<b>1</b>		<b>4</b>	<b>5</b>	<b>8</b>				<b>3</b>
			<b>7</b>		<b>6</b>			
<b>7</b>				<b>3</b>	<b>4</b>	<b>1</b>		<b>5</b>
					<b>9</b>		<b>8</b>	<b>1</b>
<b>9</b>			<b>3</b>					
<b>4</b>				<b>5</b>	<b>7</b>	<b>2</b>		<b>6</b>

(a) Opgave

<b>3</b>	<b>2</b>	<b>9</b>	<b>8</b>	<b>7</b>	<b>1</b>	<b>5</b>	<b>6</b>	<b>4</b>
<b>6</b>	<b>4</b>	<b>1</b>	<b>2</b>	<b>9</b>	<b>5</b>	<b>7</b>	<b>3</b>	<b>8</b>
<b>8</b>	<b>7</b>	<b>5</b>	<b>4</b>	<b>6</b>	<b>3</b>	<b>9</b>	<b>1</b>	<b>2</b>
<b>1</b>	<b>9</b>	<b>4</b>	<b>5</b>	<b>8</b>	<b>2</b>	<b>6</b>	<b>7</b>	<b>3</b>
<b>5</b>	<b>3</b>	<b>2</b>	<b>7</b>	<b>1</b>	<b>6</b>	<b>8</b>	<b>4</b>	<b>9</b>
<b>7</b>	<b>6</b>	<b>8</b>	<b>9</b>	<b>3</b>	<b>4</b>	<b>1</b>	<b>2</b>	<b>5</b>
<b>2</b>	<b>5</b>	<b>7</b>	<b>6</b>	<b>4</b>	<b>9</b>	<b>3</b>	<b>8</b>	<b>1</b>
<b>9</b>	<b>1</b>	<b>6</b>	<b>3</b>	<b>2</b>	<b>8</b>	<b>4</b>	<b>5</b>	<b>7</b>
<b>4</b>	<b>8</b>	<b>3</b>	<b>1</b>	<b>5</b>	<b>7</b>	<b>2</b>	<b>9</b>	<b>6</b>

(b) Oplossing

Figuur 1.2: Een Sudoku-puzzel

# Hoofdstuk 2

## Binary Decision Diagrams

### 2.1 BDDs in propositiologica

#### 2.1.1 Binary Decision Trees

We kennen verschillende normaalvormen voor een formule. De meest gekende en meest gebruikte normaalvormen zijn de conjunctieve en de disjunctieve normaalvorm (CNF en DNF)<sup>1</sup>. Gegeven een propositionele formule  $F$ , willen we vaak weten of  $F$  een tautologie<sup>1</sup> is, en zo nee, of  $F$  dan satisfieerbaar<sup>1</sup> is. Het oplossen van deze problemen is niet altijd zo vanzelfsprekend. Voor een formule in conjunctieve normaalvorm is het beslissen of de formule een tautologie is, mogelijk in polynome tijd, maar beslissen of de formule satisfieerbaar is, is een NP-compleet probleem ([Coo71]). Voor een formule in disjunctieve normaalvorm is het niet veel beter, daar is het beslissen van de satisfieerbaarheid mogelijk in polynome tijd, maar beslissen dat de formule een tautologie is, is voor zo'n formule dan weer een co-NP-compleet probleem ([Coo71]).

**Definitie 2.1.** *(We schrijven  $a \rightarrow b, c$  als afkorting voor  $(a \wedge b) \vee (\neg a \wedge c)$ , en we kunnen het lezen als If  $a$  Then  $b$  Else  $c$ .) Een formule in propositiologica in **If-Then-Else Normaalvorm (ITE-normaalvorm)** over een vocabularium  $\Sigma$ , is van de vorm:*

$$\begin{array}{ll} \phi, \psi := & \mathbf{f} \qquad \qquad \qquad (false) \\ & \mathbf{t} \qquad \qquad \qquad (true) \\ & x \rightarrow \phi, \psi \qquad \qquad (waarbij x \in \Sigma) \end{array}$$

Voor we deze vorm een normaalvorm mogen noemen, moeten we natuurlijk wel bewijzen dat elke formule in deze vorm geschreven kan worden.

**Propositie 2.1.** *Elke propositionele formule kan geschreven worden in ITE-normaalvorm.*

---

<sup>1</sup>Voor de definitie zie hoofdstuk 1.1



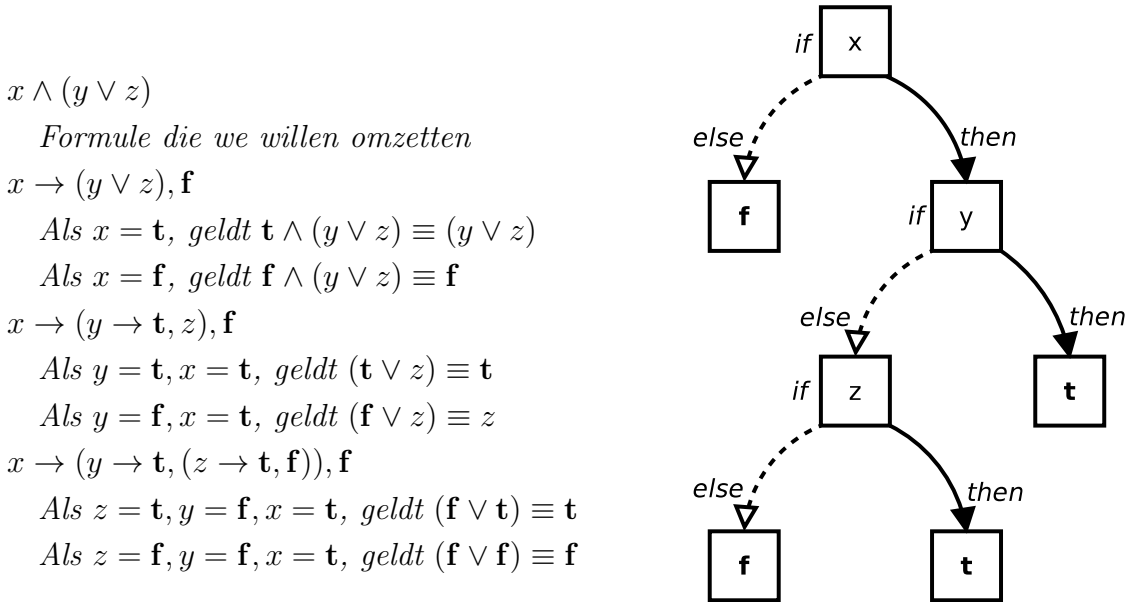
*Bewijs.* Neem een formule  $F$ , we bewijzen de propositie per inductie op het aantal variabelen in  $F$ . Stel dus om te beginnen dat  $F$  geen variabelen bevat, dan is  $F$  equivalent met een formule van de vorm  $\mathbf{f}$  of  $\mathbf{t}$ , en dus triviaalwijze in ITE-normaalvorm. Stel dan dat  $F$  meerdere variabelen bevat, en zij  $x$  één van die variabelen. Dan is het duidelijk dat  $F$  equivalent is met

$$x \rightarrow F[\mathbf{t}/x], F[\mathbf{f}/x], \quad (2.1)$$

waar me met  $F[a/x]$ ,  $F$  met  $x$  vervangen door  $a$  bedoelen. Nu geldt dat  $F[\mathbf{t}/x]$  en  $F[\mathbf{f}/x]$  minder variabelen bevatten dan  $F$ , en dus wegens de inductiehypothese in ITE-normaalvorm te schrijven zijn. Hier volgt rechtstreeks uit wat we moesten bewijzen, namelijk dat  $F$  in ITE-normaalvorm te schrijven is.  $\square$

Merk op dat dit bewijs ook een methode geeft om deze vorm te vinden. Vergelijking (2.1) noemen we de **Shannon uitbreiding** ([Bry86]) van  $F$  voor  $x$ .

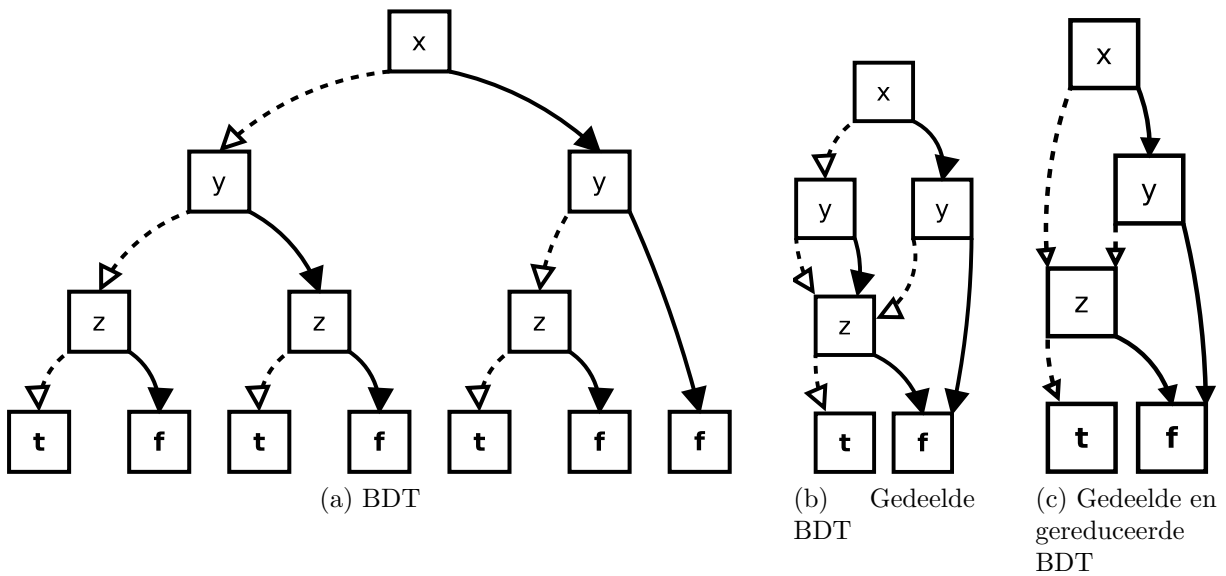
Deze INF-normaalvorm, bevat duidelijk een binaire boomstructuur. We kunnen een formule in INF-normaalvorm  $x \rightarrow s, t$  ( $s, t$  termen in INF-normaalvorm) als een boom voorstellen, waarbij  $x$  de wortel is, en  $s$  en  $t$  de 2 kinderen. Deze binaire boomstructuren noemen we **binaire beslissingsbomen** (BDT, “Binary Decision Trees”). Ter verduidelijking wordt het omzetten naar een INF-normaalvorm en het tekenen van een BDT met een voorbeeld geïllustreerd in figuur 2.1. In de tekening zien we dat we de “else-tak” (of falsetak) voorstellen met een stippellijn, en de “then-tak” (of truetak) met een volle lijn. Dit doen we per conventie, voor de duidelijkheid.



Figuur 2.1: Het opbouwen van een BDT voor  $x \wedge (y \vee z)$

### 2.1.2 Van BDT naar BDD

Hoewel de weergave van een formule zoals in figuur 2.1 erg overzichtelijk is voor mensen, is deze minder ideaal voor een computer. Als je een BDT in het geheugen van een computer moet opslaan, kan deze erg veel geheugen innemen, hoewel dit vaak niet nodig is. Als de boom 2 deelbomen heeft, die isomorf zijn, dan is het voor een computer veel efficiënter qua geheugengebruik, als we deze deelboom maar één keer opslaan in het geheugen. In plaats van de boom te kopiëren, kunnen we er dan twee keer naar verwijzen. Dat dit veel geheugen kan besparen zien we bijvoorbeeld in figuur 2.2. In figuur 2.2a staat een grote BDT, terwijl de boom in figuur 2.2b dezelfde formule voorstelt, maar al duidelijk veel kleiner is. Een BDT zoals die in figuur 2.2b die geen twee verschillende deelbomen heeft die isomorf zijn, noemen we een gedeelde BDT.



Figuur 2.2: Het delen en reduceren van BDT's

We zien nu in figuur 2.2b de deelboom  $y \rightarrow (z \rightarrow t, f), (z \rightarrow t, f)$  staan, en het is duidelijk dat deze formule equivalent is met,  $z \rightarrow t, f$ . We kunnen deze boom dus verder vereenvoudigen door  $y \rightarrow (z \rightarrow t, f), (z \rightarrow t, f)$  te vervangen door  $z \rightarrow t, f$ , en zo krijgen we de boom in figuur 2.2c. Deze is weer kleiner, en overzichtelijker. Deze aanpak kunnen we rechtstreeks veralgemenen voor elke knoop met 2 identieke kinderen in een boom. Stel dat we een boom hebben met wortel  $x$ , van de vorm  $x \rightarrow \psi, \psi$  (waarbij  $\psi$  ook een BDT voorstelt), dan kunnen we deze boom vervangen door de BDT  $\psi$ . Een BDT die geen knopen bevat met identieke kinderen, noemen we een gereduceerde BDT.

Deze twee opmerkingen brengen ons bijna bij de definitie van een BDD. We hebben nog één notie nodig, voor we BDDs kunnen definiëren. We introduceren eerst nog het concept van een ordening op BDT's. Met een geordende BDT, bedoelen we een BDT waar er een ordening op de knopen bestaat, waarbij  $x < y$  betekent dat  $x$  altijd boven  $y$  staat in de boom. We definiëren nu een BDD als een gedeelde, gereduceerde en geordende BDT. Meer formeel komen we tot deze definitie ([And97]):

**Definitie 2.2.** Een BDD is een gerichte acyclische graaf, met 1 initiële, en 1 of 2 terminale knopen, waarvoor geldt dat:

- De terminale knopen hebben labels(=naamgeving van de knopen) **f** en/of **t**, en de andere knopen hebben een variabele als label.
- Alle knopen buiten de terminale knopen hebben 2 uitgaande bogen, met elk een label. Eén van de bogen heeft een then-label, terwijl de andere een else-label heeft.
- Er bestaat een lineaire orde  $x_1 < x_2 < \dots < x_n$  op de variabelen, die op elk pad doorheen de graaf gerespecteerd wordt.
- Er bestaan geen 2 knopen die dezelfde variabele als label hebben, en dezelfde kinderen hebben.
- Er bestaat geen knoop die 2 identieke kinderen heeft.

Merk op dat de beperkingen die we op de BDT's hebben doorgevoerd om BDDs te verkrijgen niets veranderen aan de expressiviteit ervan, wat geeft dat elke propositionele formule door een BDD weergegeven kan worden. Deze bewering wordt bewezen in stelling 2.1. De beperkingen hebben daarentegen zelfs een heel interessant gevolg, het blijkt dat BDDs een canonieke vorm zijn voor propositielogica:

**Stelling 2.1** ([And97]). Voor elke propositionele formule  $\varphi$  bestaat er exact één equivalente BDD met variabele-ordening  $x_1 < x_2 < \dots < x_n$ .

*Bewijs.* We bewijzen deze stelling per inductie op het aantal variabelen in  $\varphi$ . Als  $\varphi$  geen variabelen bevat, zijn er 2 mogelijkheden, ofwel is  $\varphi$  de constante formule *true* en is **t** de enige BDD die  $\varphi$  voorstelt, ofwel is  $\varphi$  de constante formule *false*, en **f** de enige BDD die  $\varphi$  voorstelt.

Stel dan dat de stelling geldt voor  $n$  variabelen. Neem dan  $\varphi$  een formule met  $n + 1$  variabelen  $x_1 < x_2 < \dots < x_n < x_{n+1}$ , dan moeten we bewijzen dat er een unieke BDD bestaat die  $\varphi$  voorstelt. Definieer nu  $\psi_0 = \varphi[\mathbf{f}/x_1]$  en  $\psi_1 = \varphi[\mathbf{t}/x_1]$ . Er bestaan dan wegens de inductiehypothese unieke BDDs  $B_0$  en  $B_1$  die respectievelijk  $\psi_0$  en  $\psi_1$  weergeven.

Stel om te beginnen dat  $B_0 = B_1$ , dan geldt dat  $B = B_0 = B_1$  de enige BDD is die  $\varphi$  weergeeft. Inderdaad, stel dat er een andere BDD  $C$  bestaat die  $\varphi$  weergeeft. Als  $C$  de variabele  $x_1$  niet bevat, dan is  $C$  ook een BDD voor  $\psi_0$ . Wegens de inductiehypothese geldt dus dat  $B = C$ . Als  $C$  de variabele  $x_1$  wel bevat, moet deze vanwege de ordening helemaal van boven in de wortel staan. Dan geldt dat de truetak onder deze wortel een BDD is voor  $\varphi[\mathbf{t}/x_1]$ , en dat deze truetak dus  $B_1$  is. Analoog geldt dat de falsetak dan de BDD  $B_2$  is. Hieruit volgt dan dat de BDD  $C$  niet gereduceerd is, aangezien de true- en de falsetak onder de wortel gelijk zijn. Dit kan natuurlijk niet,  $C$  kan dus  $x_1$  niet bevatten, en er moet gelden dat  $B=C$ .

Neem nu aan dat  $B_0 \neq B_1$ , dan moet gelden dat  $\varphi$  afhankelijk is van  $x_1$ , en dat elke BDD die  $\varphi$  voorstelt dus de variabele  $x_1$  moet bevatten. Uit de ordening op de variabelen volgt

dan dat die BDD dus de variabele  $x_1$  als wortel heeft. Analoog aan hierboven kunnen we dan besluiten dat de truetak van  $x_1$  dan  $B_1$ , en de falsetak  $B_0$  moet zijn. Dit wil dus zeggen dat onze wortel eenduidig vastligt, en de twee takken van deze knoop liggen evenzeer eenduidig vast wegens de inductiehypothese. Dit bewijst dan de stelling.  $\square$

Deze stelling heeft nog een zeer interessant gevolg. Er bestaat namelijk maar één BDD die een tautologie voorstelt, namelijk de BDD  $\mathbf{t}$ . Als we dus een formule  $\varphi$  met BDD  $B$  hebben, die niet  $\mathbf{f}$  is, weten we dus al dat  $\varphi$  satisfieerbaar is, aangezien een formule  $\psi$  satisfieerbaar is, als en slechts als  $\neg\psi$  geen tautologie is.

## 2.2 BDDs voor Predicatenlogica

### 2.2.1 FOBDDs

We veralgemenen nu de notie van BDDs voor formules in eerste-orde logica. We moeten dus onze BDDs zo aanpassen, dat er met kwantoren kan omgegaan worden. We gaan dit proberen op een natuurlijke manier uit te breiden, maar dan moeten we wel een oplossing vinden voor het gebruik van kwantoren. We gaan de kwantoren eigenlijk behandelen als “black box”, met daarin een BDD voor de formule in de scope van de kwantor. Om dit duidelijker te maken hebben we eerst notie van kernen nodig:

**Definitie 2.3.** Een **kern** is een formule in eerste-orde logica, van volgende vorm:

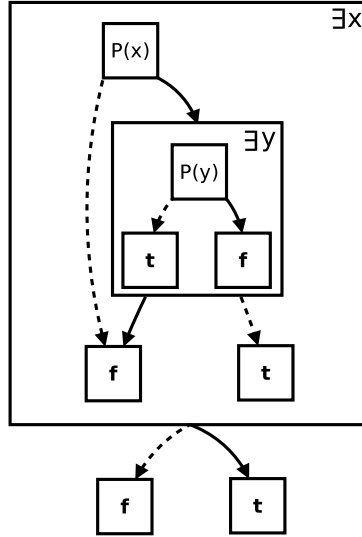
$$\begin{aligned} K &:= \phi && \text{met } \phi \text{ een atomische formule} \\ &\quad \exists x(\psi) && \text{met } x \text{ een variabele, en } \psi \in \mathcal{L}_{\mathbf{FO}} \end{aligned}$$

We merken op dat we een eerste-orde formule kunnen definiëren als een propositionele combinatie van kernen, als we eerst elke deelformule van de vorm  $\forall x(\phi)$  vervangen door  $\neg(\exists x(\neg\phi))$ .

Deze definitie maakt al veel duidelijker wat we bedoelen, we kunnen nu een duidelijk verband zien tussen propositionele formules en eerste-orde formules gedefinieerd met kernen. Een propositionele formule is een propositionele combinatie van propositionele variabelen, en een eerste-orde formule is een propositionele combinatie van kernen.

Stel nu dat we een eerste-orde formule  $\phi$  hebben, waar een FOBDD voor willen opstellen, dan kunnen we elke kern vervangen door een propositionele variabele  $K_i$ . Dan hebben we dus wegens de opmerking hierboven een propositionele formule, waar we een BDD voor kunnen opstellen. In die BDD gaan we dan alle propositionele variabelen terug proberen om te zetten naar de bijhorende kern. De kern horende bij  $K_i$  heeft één van de volgende twee vormen:

- Het is een atomische kern, dan kunnen we  $K_i$  vervangen door deze atomische formule.



Figuur 2.3: EFOBDD voor  $\exists x(P(x) \wedge \forall y(P(y)))$

- Het is een kwantificatiekern, dan moeten we eerst op analoge manier een BDD opstellen voor een formule binnen de kwantificatie. Als dit gebeurd is, hebben we een FOBDD voor de formule binnen de kwantificatie. Hier zetten we tenslotte terug een kwantificatie rond (zie figuur 2.3), om de uiteindelijke knoop te krijgen zoals die in onze FOBDD zal staan.

We definiëren een eerste-orde-BDD formeel:

**Definitie 2.4.** Een *eerste-orde-BDD* (FOBDD) is een gerichte acyclische graaf, met 1 initiële, en 1 of 2 terminale knopen, waarvoor geldt dat:

- De terminale knopen hebben labels **f** en/of **t**.
- Alle knopen buiten de terminale knopen hebben 2 uitgaande bogen.
- Alle knopen buiten de terminale knopen hebben als label ofwel een atomische formule, ofwel een kwantificatiekern.
- Er bestaat een lineaire orde  $K_1 < K_2 < \dots < K_n$  op de kernen, die op elk pad doorheen de graaf gerespecteerd wordt.
- Er bestaan geen 2 knopen die hetzelfde label, en dezelfde kinderen hebben.
- Er bestaat geen knoop die 2 identieke kinderen heeft.
- Bij alle knopen waar het label bestaat uit een kwantificatiekern, is het label de vorm  $\exists x(\phi)$ , waarbij  $\phi$  zelf een eerste-orde-BDD voorstelt.

Merk op dat we een FOBDD of een knoop hiervan nog steeds als een formule in eerste-orde logica kunnen beschouwen, we kunnen namelijk elke FOBDD één op één mappen op

een geordende en gereduceerde eerste-orde BDT (FOBDT)<sup>2</sup>, en een FOBDT is een formule in eerste-orde logica. Meer nog, een FOBDT is een normaalvorm voor eerste-orde logica, zoals blijkt uit volgend resultaat:

**Propositie 2.2.** *Voor elke eerste-orde formule  $\varphi$  bestaat een equivalente FOBDD.*

*Bewijs.* We bewijzen deze propositie per inductie op het aantal kwantoren. Als  $\varphi$  geen kwantoren bevat, is het dus een propositionele formule, en volgt het te bewijzen dus rechtstreeks uit propositie 2.1. Stel dus dat  $\varphi$   $n$  kwantoren bevat, dan kunnen we  $\varphi$  schrijven als een booleaanse combinatie van kernen, wegens de opmerking in definitie 2.3. Noem deze kernen  $\varphi_1, \dots, \varphi_k$ . Stel zonder verlies van algemeenheid dat  $\varphi_1, \dots, \varphi_{i-1}$  atomische formules zijn, en dat  $\varphi_i, \dots, \varphi_k$  gekwantificeerde formules zijn. Schrijf dan dat

$$\begin{aligned}\varphi_i &= (\exists x(\psi_i)) \\ &\dots \\ \varphi_k &= (\exists x(\psi_k))\end{aligned}$$

Dan geldt dat alle  $\psi_j$  ( $j = i..k$ ) maximaal  $n - 1$  kwantoren bevatten, en er dus voor elke  $\psi_j$  een equivalente FOBDD  $B_j$  bestaat.

Hieruit volgt dan dat we elke  $\varphi_i, \dots, \varphi_k$  kunnen vervangen door een knoop met een kwantificatiekern als label, en elke  $\varphi_1, \dots, \varphi_{i-1}$  door een knoop met een atomische formule als label. Gebruikmakend van de Shannon-uitbreiding in propositie 2.1 kunnen we dan een FOBDT voor  $\varphi$  opstellen, en deze vervolgens vereenvoudigen tot een FOBDD.  $\square$

Jammer genoeg kunnen we geen veralgemening formuleren van stelling 2.1. Dit is natuurlijk geen verrassing, aangezien dit zou betekenen dat we beslissingsprocedure zouden gevonden hebben voor de volledige eerste-orde logica. Dit kan natuurlijk niet, aangezien eerste-orde logica onbeslisbaar is ([BBJ02]). We zien wel in de praktijk wel dat in veel gevallen toch vaak dezelfde bomen zien voor equivalente formules. We willen dit ook zo goed mogelijk benaderen, omdat dit één van de interessante punten aan BDDs was. We gaan dieper in op dit probleem in hoofdstuk 3.2.

## 2.2.2 Opstellen van FOBDDs in de praktijk

De manier waarop we BDDs geïntroduceerd hebben in hoofdstuk 2.1.2 (meer specifiek in propositie 2.2), geeft ons ook een algoritme om een eerste-orde formule te vertalen naar een equivalente FOBDD. Maar dit is in de praktijk echter geen goed idee. Dit zou betekenen dat we voor elke formule eerst een FOBDT maken, waarin we dan onder andere alle paren deelbomen zouden moeten vergelijken, om gelijke deelbomen te kunnen vinden. Een efficiënter plan is om de FOBDDs bottom-up op te bouwen. Hiervoor hebben we eerst en vooral volgende eigenschap nodig:

---

<sup>2</sup>We kunnen een FOBDT gelijkaardig definiëren als een FOBDD, namelijk als een BDT met atomische formules of existentieel gekwantificeerde BDT's als knopen.

**Propositie 2.3.** *Gegeven twee BDDs  $B_1 = \xi \rightarrow \alpha, \beta$ ,  $B_2 = \chi \rightarrow \delta, \varepsilon$  en  $B_3 = \xi \rightarrow \eta, \kappa$ , waarvoor  $\xi < \chi$  dan geldt:*

$$\neg \mathbf{t} \equiv \mathbf{f} \quad (2.2)$$

$$\neg \mathbf{f} \equiv \mathbf{t} \quad (2.3)$$

$$\neg B_1 \equiv \xi \rightarrow \neg \alpha, \neg \beta \quad (2.4)$$

$$\exists x : B_1 \equiv \exists x[\xi \rightarrow \alpha, \beta] \rightarrow \mathbf{t}, \mathbf{f} \quad (2.5)$$

$$\forall x : B_1 \equiv \exists x[\xi \rightarrow \neg \alpha, \neg \beta] \rightarrow \mathbf{f}, \mathbf{t} \quad (2.6)$$

$$B_1 \wedge B_2 \equiv \xi \rightarrow (\alpha \wedge B_2), (\beta \wedge B_2) \quad (2.7)$$

$$B_1 \vee B_2 \equiv \xi \rightarrow (\alpha \vee B_2), (\beta \vee B_2) \quad (2.8)$$

$$B_1 \wedge B_3 \equiv \xi \rightarrow (\alpha \wedge \eta), (\beta \wedge \kappa) \quad (2.9)$$

$$B_1 \vee B_3 \equiv \xi \rightarrow (\alpha \vee \eta), (\beta \vee \kappa) \quad (2.10)$$

*Bewijs.* Gevallen (2.2), (2.3), (2.5) en (2.6) zijn triviaal, en de bewijzen voor gevallen (2.8), (2.9) en (2.10) zijn allemaal analoog aan het bewijs voor (2.7).

We bewijzen dus enkel (2.4) en (2.7).

- (2.4):

$$\neg B_1 \equiv \neg(\xi \rightarrow \alpha, \beta) \quad (2.11)$$

$$\equiv \neg((\xi \wedge \alpha) \vee (\neg \xi \wedge \beta)) \quad (2.12)$$

$$\equiv ((\neg \xi \vee \neg \alpha) \wedge (\xi \vee \neg \beta)) \quad (2.13)$$

$$\equiv ((\neg \xi \vee \neg \alpha) \wedge (\xi \vee \neg \beta) \wedge (\neg \alpha \vee \neg \beta)) \quad (2.14)$$

$$\equiv ((\xi \vee \neg \xi) \wedge (\neg \xi \vee \neg \alpha) \wedge (\xi \vee \neg \beta) \wedge (\neg \alpha \vee \neg \beta)) \quad (2.15)$$

$$\equiv ((\xi \wedge \neg \alpha) \vee (\neg \xi \wedge \neg \beta)) \quad (2.16)$$

$$\equiv \xi \rightarrow \neg \alpha, \neg \beta \quad (2.17)$$

Waarbij (2.14) geldt omdat  $(\neg \xi \vee \neg \alpha) \wedge (\xi \vee \neg \beta) \Rightarrow (\neg \alpha \vee \neg \beta)$

- (2.7): We gaan bewijzen dat  $B_1 \wedge B_2$  equivalent is met  $\xi \rightarrow (\alpha \wedge B_2), (\beta \wedge B_2)$ . We bewijzen eerst dat  $B_1 \wedge B_2 \Rightarrow \xi \rightarrow (\alpha \wedge B_2), (\beta \wedge B_2)$ . Stel dus dat  $B_1 \wedge B_2$  waar is, dan zijn  $B_1$  en  $B_2$  dus beide waar.

- Als dan  $\xi$  waar is, dan geldt ook  $\alpha$ , en dus ook  $\alpha \wedge B_2$ . Hieruit volgt dan dat natuurlijk dat  $\xi \rightarrow (\alpha \wedge B_2), (\beta \wedge B_2)$  geldt.
- Analoog, als  $\xi$  onwaar is, dan geldt dat  $\beta$ , en dus ook  $\beta \wedge B_2$  waar is. Hieruit volgt dan dat opnieuw dat  $\xi \rightarrow (\alpha \wedge B_2), (\beta \wedge B_2)$  geldt.

Stel nu dat  $\xi \rightarrow (\alpha \wedge B_2), (\beta \wedge B_2)$  waar is.

- Als  $\xi$  waar is, geldt  $\alpha \wedge B_2$ . Omdat  $\alpha \wedge \xi$  geldt, en  $B_2$  waar is volgt dat  $B_1 \wedge B_2$  geldt.
- Als  $\xi$  onwaar is, geldt  $\beta \wedge B_2$ . Omdat  $\beta \wedge \neg \xi$  geldt, en  $B_2$  waar is volgt dat  $B_1 \wedge B_2$  geldt.

□

Gebruikmakend van deze eigenschap kunnen we dus een FOBDD bottom-up opstellen. We maken eerst voor elke atomische formule  $A$  een FOBDD  $A \rightarrow \mathbf{t}, \mathbf{f}$ , en gebruiken dan propositie 2.3 om onze FOBDD van binnenuit op te stellen. Telkens we een nieuwe kern aanmaken, voegen we deze kern toe aan onze volgorde. Op deze manier hebben we altijd éénduidig een volgorde vastliggen. We werken dit concreet uit met een voorbeeld: het opstellen van een FOBDD voor  $x = 2 \wedge \exists a : (a + 1 < x \vee P(a))$ .

**Voorbeeld 2.1.** We willen een FOBDD opstellen voor de formule

$$x = 2 \wedge \exists a : (a + 1 < x \vee P(a))$$

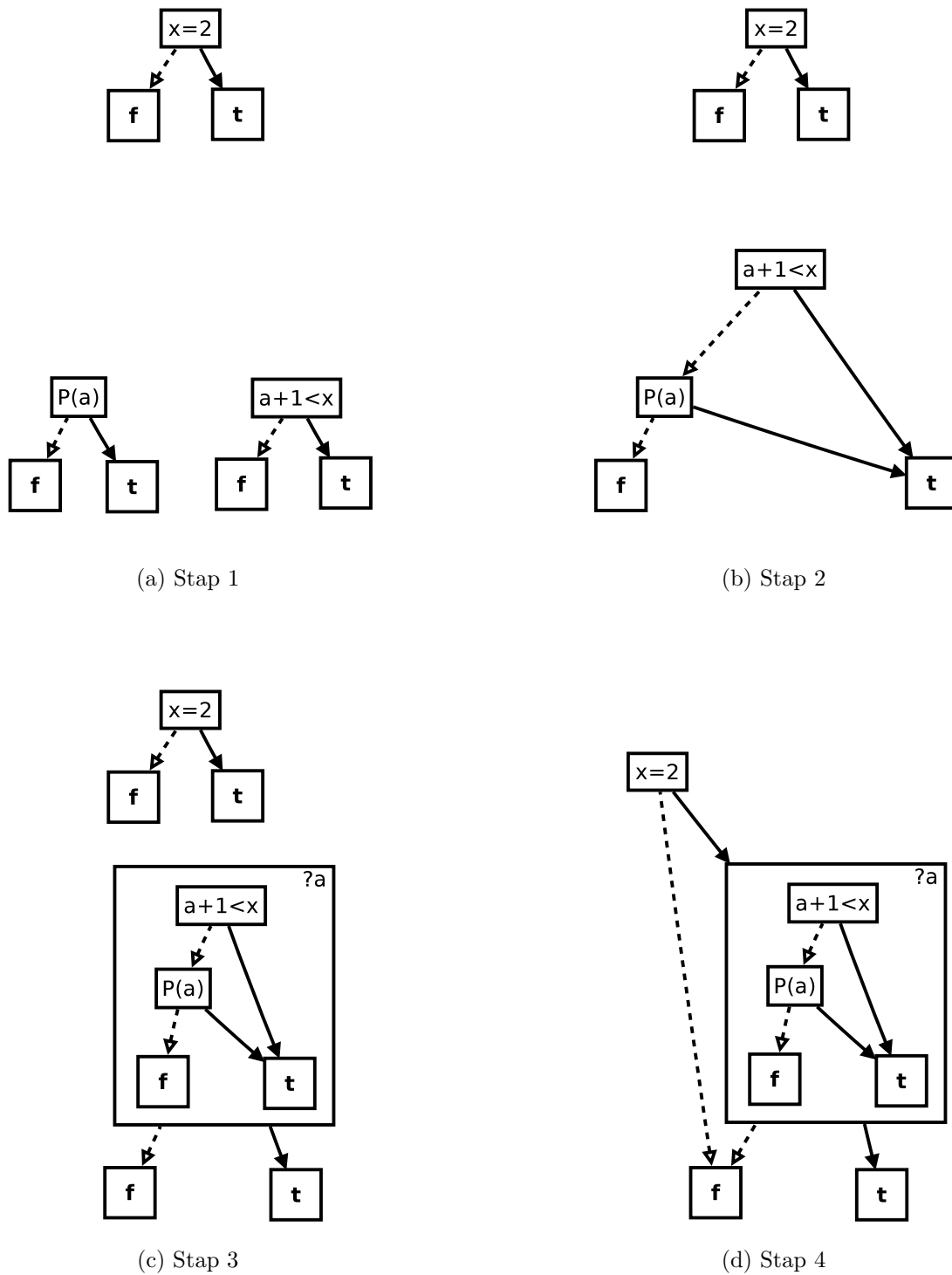
Dit doen we in vier stappen (deze stappen worden visueel weergegeven in figuur 2.4):

- In de eerste stap, maken we voor elke atomische formule  $A$  een FOBDD  $A \rightarrow \mathbf{t}, \mathbf{f}$ . We bekomen zo dus drie FOBDDs. Dit is er dan één voor  $x = 2$ , één voor  $P(a)$  en één voor  $a + 1 < x$ . We kiezen daarbij ook meteen de volgorde, namelijk dat  $\text{volgnr}(x = 2) < \text{volgnr}(a + 1 < x) < \text{volgnr}(P(a))$ . Merk op dat we in figuur 2.4a, de true- en falseknoop drie keer tekenen, maar dit is enkel voor overzichtelijkheid, er is natuurlijk maar één trueknoop en één falseknoop.
- We stellen nu de FOBDD op voor  $a + 1 < x \vee P(a)$ . We zien dat in onze volgorde  $a + 1 < x$  boven  $P(a)$  moet staan. We krijgen dus dat

$$\begin{aligned} & (a + 1 < x \rightarrow \mathbf{t}, \mathbf{f}) \vee (P(a) \rightarrow \mathbf{t}, \mathbf{f}) \\ &= a + 1 < x \rightarrow (\mathbf{t} \vee (P(a) \rightarrow \mathbf{t}, \mathbf{f})), (\mathbf{f} \vee (P(a) \rightarrow \mathbf{t}, \mathbf{f})) \\ &= a + 1 < x \rightarrow (\mathbf{t}), ((P(a) \rightarrow \mathbf{t}, \mathbf{f})) \end{aligned}$$

- We maken nu een nieuwe kwantificatiekern voor  $\exists a : (a + 1 < x \vee P(a))$ . Dit betekent ook dat we deze toevoegen aan onze volgorde, en dit helemaal als laatste.
- Nu moeten we enkel nog de conjunctie maken van de nieuwe kwantificatiekern met  $x = 2 \rightarrow \mathbf{t}, \mathbf{f}$ . Dit is analoog met de eerste stap, met dan natuurlijk de opmerking dat  $\varphi \wedge \mathbf{t}$  equivalent is met  $\varphi$  en  $\varphi \wedge \mathbf{f}$  met  $\mathbf{f}$  voor elke formule  $\varphi$ .




 Figuur 2.4: Het Bottom-Up opbouwen van een FOBDD voor  $x = 2 \wedge \exists a : (a + 1 < x \vee P(a))$

# Hoofdstuk 3

## Kennisrepresentatie en het IDP-systeem

### 3.1 Modelgeneratie en -expansie

#### 3.1.1 Werking van modelexpansie

In hoofdstuk 1.2 hebben we de definities van modelgeneratie en -expansie al gezien. We bespreken hier hoe we een modelexpansie<sup>1</sup> kunnen uitvoeren. Gegeven twee vocabularia  $\sigma, \Sigma$ , waarbij  $\sigma \subset \Sigma$ , een theorie  $\mathcal{T}$  over  $\Sigma$  en een eindige interpretatie  $I$  van  $\sigma$ , zoeken we een model  $M$  voor  $\mathcal{T}$  over  $\Sigma$ , zodat de beperking van  $M$  tot  $\sigma$  gelijk is aan  $I$ . Het uitvoeren van een modelexpansie kunnen we beschrijven in drie stappen, die we hier kort overlopen. We gebruiken een simpel voorbeeld om de drie stappen te kunnen illustreren.

**Voorbeeld 3.1.** We voeren een modelexpansie uit voor  $\Sigma, \sigma, \mathcal{T}$  en  $I$  waarbij

$$\Sigma = \{\{N\}, \{P(N), Q(N)\}, \{\}\}$$

$$\sigma = \{\{N\}, \{P(N)\}, \{\}\}$$

$$\mathcal{T} = \{\forall x(\forall y(P(x) \vee Q(y)))\}$$

$$N^I = \{0, 1\}$$

$$P^I = \{0\}$$

De drie stappen voor een modelexpansie zijn dan:

1. In de eerste stap wordt een grounding voor  $\mathcal{T}$  opgesteld (hoofdstuk 3.1.3). Dit wil zeggen dat we  $\mathcal{T}$  met eindig domein  $D$ <sup>2</sup> omzetten naar een equivalente propositionale theorie  $\mathcal{T}'$ . We stellen  $\mathcal{T}'$  zo op, zodat alle formules in  $\mathcal{T}'$  in conjunctieve normaalvorm staan.

---

<sup>1</sup>En bijgevolg ook een modelgeneratie, aangezien dit een modelexpansie is met lege inputstructuur.

<sup>2</sup>Bij het uitvoeren kennen we het domein (oftewel de interpretatie) voor alle types, wegens de opmerking na definitie 1.16.

**Voorbeeld 3.1b.** De formule  $\forall x(\forall y(P(x) \vee Q(y))) \in \mathcal{T}$  met domein  $D_N = \{0, 1\}$  kunnen we omzetten naar:

$$(P(0) \vee Q(0)) \wedge (P(0) \vee Q(1)) \wedge (P(1) \vee Q(0)) \wedge (P(1) \vee Q(1)) \quad (3.1)$$

2. Als tweede stap kunnen we vervolgens hierop onze interpretatie  $I$  toepassen, en de propositiesymbolen waar we een interpretatie van kennen vervangen door **t** of **f**, waarna we de theorie kunnen vereenvoudigen. Dit doen we per formule, door volgende gekende eigenschappen herhaaldelijk toe te passen:

- $\varphi \vee \mathbf{t} \equiv \mathbf{t}$
- $\varphi \vee \mathbf{f} \equiv \varphi$
- $\varphi \wedge \mathbf{t} \equiv \varphi$
- $\varphi \wedge \mathbf{f} \equiv \mathbf{f}$

**Voorbeeld 3.1c.** Gegeven formule  $N^I := \{0, 1\}$  en  $P^I := \{0\}$ , kunnen we formule (3.1) omzetten naar:

$$((\mathbf{t} \vee Q(0)) \wedge (\mathbf{t} \vee Q(1))) \wedge (\mathbf{f} \vee Q(0)) \wedge (\mathbf{f} \vee Q(1)) \quad (3.2)$$

En deze formule kunnen we reduceren tot:

$$Q(0) \wedge Q(1) \quad (3.3)$$

3. De laatste stap, is dan deze nieuwe theorie aan een SAT-solver meegeven, welke dan de satisfieerbaarheid van deze theorie bepaalt, en een oplossing geeft in het geval de theorie satisfieerbaar is.

**Voorbeeld 3.1** De formule  $Q(0) \wedge Q(1)$  is duidelijk satisfieerbaar, namelijk met de interpretatie  $Q(0) = Q(1) = \mathbf{t}$ . Dit geeft ons dus een model  $M$  voor  $\Sigma$ , zodat  $N^M = \{0, 1\}$ ,  $P^M = \{0\}$  en  $Q^M = \{0, 1\}$ .

Het is duidelijk dat de tweede stap niet zo tijdsconsumerend is. Het meeste werk/tijd kruipt in het grounden, en het oplossen van het SAT-probleem voor de equivalente propositionale theorie. Als we de efficiëntie van het systeem willen verbeteren, is het dus belangrijk om dit te doen bij deze twee processen. In de context van deze thesis wordt er enkel gekeken naar een verbetering in de werking van het groundingsalgoritme.

De manier waarop de modelexpansie hierboven uitgewerkt werd, is de theoretische aanpak, in de praktijk gebeurt het ongeveer analoog. De manier waarop het IDP-systeem<sup>3</sup> de modelexpansie uitvoert is eigenlijk een optimalisatie van dit algoritme. In IDP gebeuren bijvoorbeeld stap twee en stap drie tegelijk om de grounding efficiënter te maken.

---

<sup>3</sup>zie hoofdstuk 3.1.2

### 3.1.2 IDP

In hoofdstuk 1.2, werd gesproken over het vakgebied van de kennisrepresentatie, waarin kennis kan gegeven worden aan een computer, die er dan mee kan redeneren. Een computerprogramma hiervoor noemen we een KBS (“knowledge-based system”). Een KBS is een systeem dat kennis over een bepaald domein kan opslaan en gebruiken om allerlei opdrachten uit te voeren waarvoor de kennis nodig is, en allerlei conclusies kan trekken uit deze kennis.

Het IDP-systeem ([MWD06]) is een KBS dat onder meer bruikbaar is voor het uitvoeren van modelexpansie. Het systeem wordt ontwikkeld door de onderzoeksgroep KRR (Knowledge Representation and Reasoning) van de afdeling DTAI (Declaratieve Talen en Artificiele Intelligentie) op het departement computerwetenschappen aan de KU Leuven. Er wordt gebruik gemaakt van FO(.) als invoertaal, een uitbreiding van eerste-orde logica, die ook in deze onderzoeksgroep ontwikkeld werd (zie hoofdstuk 1.1.3). De manier van invoeren is heel gelijkaardig aan de manier waarop wij de problemen in eerste-orde logica gespecificeerd hebben in hoofdstuk 1.2.1. Je definieert een vocabularium, een theorie met dat vocabularium en de logische symbolen uit FO(.) en een structuur. Het systeem probeert dan deze structuur uit te breiden tot een volledige structuur voor de theorie. Bij wijze van illustratie zullen we hier het voorbeeld van de sudoku uitschrijven in de invoertaal voor IDP. IDP gebruikt een ASCII-versie van de syntax voor eerste orde logica, een korte vertaling is te vinden in tabel 3.1<sup>4</sup>.

FO	IDP	FO	IDP
$\neg$	$\sim$	$\Leftrightarrow$	$<=>$
$\wedge$	$\&$	$\forall$	$!$
$\vee$	$ $	$\exists$	$?$
$\Rightarrow$	$=>$	$=$	$=$
$\Leftarrow$	$<=$	$\neq$	$\sim=$

Tabel 3.1: Vertaling FO naar ASCII

**Voorbeeld 3.2.** Sudoku in de IDP-invoertaal:

```

vocabulary Voc {
  type B isa nat
  type Row isa nat
  type Col isa nat
  type Num isa nat
  Given(Row, Col, Num)
  Solution(Row, Col, Num)
  Block(Row, Col):B
}

theory Sudoku : Voc {
  ! x y z : Given(x,y,z)=> Solution(x,y,z).

```

<sup>4</sup>Dit is een ASCII-versie van de syntax voor eerste-orde logica, voor meer informatie hierover zie [KUL12]

```

! x y : ?1 z : Solution(x,y,z).
! x z : ?1 y : Solution(x,y,z).
! y z : ?1 x : Solution(x,y,z).
! b z : ?1 x y : b=Block(x,y) & Solution(x,y,z).
! x y : Block(x,y)= ((x-1)/3)*3 + ((y-1)/3) + 1.
}

structure S : Voc {
  Row={1..9}
  Col={1..9}
  Num={1..9}
  B={1..9}
  Given={1,1,3;1,3,9;1,4,8;1,5,7;1,9,4;
        2,6,5;2,9,8;
        3,1,8;3,2,7;3,4,4;
        4,1,1;4,3,4;4,4,5;4,5,8;4,9,3;
        5,4,7;5,6,6;
        6,1,7;6,5,3;6,6,4;6,7,1;6,9,5;
        7,6,9;7,8,8;7,9,1;
        8,1,9;8,4,3;
        9,1,4;9,5,5;9,6,7;9,8,2;9,9,6}
}

```

Zoals eerder vermeld, is de werking van IDP bij modelexpansie gelijkaardig aan de methode die hierboven beschreven is. De theorie wordt eerst omgezet in een equivalente propositionele theorie, met eventueel een aantal extra constructies voor bijvoorbeeld inductieve definities. Vervolgens wordt de propositionele theorie meegegeven aan een extended SAT-solver. De SAT-solver die gebruikt wordt in IDP, is Minisat-ID. Dit is een variant op de SAT-solver Minisat [Een04], die uitgebreid werd om met de verschillende uitbreidingen in FO(.) te kunnen werken.

### 3.1.3 Grounden

Het grounden van een theorie  $\mathcal{T}$  in eerste-orde logica, met een eindig domein, is het omzetten naar een equivalente theorie in propositionele logica, zoals we gedaan hebben in voorbeeld 3.1b. Formeel definiëren we grounden als volgt:

**Definitie 3.1.** *Zij  $\varphi$  een zin in de eerste-orde logica, waarvoor we een eindig domein hebben voor de types van de variabelen in  $\varphi$ . Dan noemen we het omzetten van  $\varphi$  naar een equivalente propositionele formule  $\varphi'$  het **grounden** van  $\varphi$ . De formule  $\varphi'$  noemen we dan de **grounding**. De grounding van een theorie  $\mathcal{T}$  noemen we bij uitbreiding de theorie  $\mathcal{T}'$  die bestaat uit de grounding  $\varphi'$  van elke  $\varphi \in \mathcal{T}$ .*

De meest voor de hand liggende manier om een willekeurige theorie te grounden, is exact de manier waarop we dit gedaan hebben in voorbeeld 3.1b. Deze manier noemen we volledig grounden, zoals te zien in algoritme 3.1. Het nadeel van deze methode wordt

---

**Algoritme 3.1:** Volledige Grounding

---

**input** : Een zin  $\varphi$ , en een eindig domein  $D_x$  voor alle variabelen in  $\varphi$ **output**: Een propositionele zin  $\varphi'$ 

```

1 switch  $\varphi$  do
2   case  $\varphi$  is (negatie van) atomische formule
3     return  $\varphi$ 
4   case  $\varphi = \varphi_1 \vee \dots \vee \varphi_n$ 
5     return  $\text{Ground}(\varphi_1) \vee \text{Ground}(\varphi_2) \vee \dots \vee \text{Ground}(\varphi_n)$ 
6   case  $\varphi = \varphi_1 \wedge \dots \wedge \varphi_n$ 
7     return  $\text{Ground}(\varphi_1) \wedge \text{Ground}(\varphi_2) \wedge \dots \wedge \text{Ground}(\varphi_n)$ 
8   case  $\varphi = \forall x \psi[x]$ 
9      $G := \emptyset$ ;
10    for  $d \in D_x$  do
11       $G := G \cup \text{ground}(\psi[d], D)$ 
12    end
13    return  $G_1 \wedge G_2 \wedge \dots \wedge G_n$  ;           // Waarbij  $G = \{G_1, \dots, G_n\}$ 
14  case  $\varphi = \exists x \psi[x]$ 
15     $G := \emptyset$ ;
16    for  $d \in D_x$  do
17       $G := G \cup \text{ground}(\psi[d], D)$ 
18    end
19    return  $G_1 \vee G_2 \vee \dots \vee G_n$  ;           // Waarbij  $G = \{G_1, \dots, G_n\}$ 
20
21 end

```

---

snel duidelijk. Stel namelijk dat  $\varphi$  een zin is, met veel geneste kwantoren, dan wordt de grounding van  $\varphi$  een zeer lange zin. De lengte van de grounding van een formule is namelijk exponentieel in functie van de maximale diepte<sup>5</sup> van de kwantoren in de formule. Het is dus zeker de moeite om na te denken over efficiëntere manieren van grounden, want zo een enorme grounding zal ook vertraging opleveren bij het uitzoeken van de satisfieerbaarheid.

## 3.2 Verbeteringen voor een groundingsalgoritme

Bij het bespreken van algoritme 3.1 werd er opgemerkt dat dit waarschijnlijk geen optimale methode is om een grounding uit te voeren. Het is de meest voor de hand liggende, maar wel een erg naïeve methode om een grounding uit te voeren. Zeker als we herinneren dat we deze grounding uitvoeren in de context van modelexpansie, en dat we dus voor een deelverzameling  $\sigma \subset \Sigma$  van het vocabularium al een interpretatie hebben. Na het grounden wordt deze kennis ook toegepast, en zoals beschreven in hoofdstuk 3.1, maakt dit de formule terug veel kleiner. Het is dus duidelijk dat het in deze context niet zo efficiënt is, om eerst de volledige grounding te berekenen, en deze pas daarna te reduceren met behulp van de interpretatie  $I$  van  $\sigma$ . Het zou veel efficiënter zijn, om de informatie van de interpretatie tijdens het grounden te gebruiken. Zo zien we bijvoorbeeld dat het zinloos was om in voorbeeld 3.1b de grounding uit te voeren, voor alle  $y$  als  $x = 0$ , want we wisten al dat  $\forall y(P(0) \vee Q(y))$  waar zou zijn.

Het gebruiken van informatie die vooraf gekend is, moet niet alleen beperkt blijven tot het invullen van de interpretaties, en het reduceren van de formules. Stel bijvoorbeeld dat we een theorie  $\mathcal{T} = \{\forall x(P(x)), \forall x(P(x) \vee Q(x))\}$  hebben. Het is dan niet zinvol om  $\forall x(P(x) \vee Q(x))$  te grounden, aangezien deze formule volgt uit  $\forall x(P(x))$ , welke ook in de theorie zit. We kunnen op veel verschillende manieren informatie halen uit onze theorie en onze interpretatie, en we willen deze informatie graag zo goed mogelijk gebruiken. Een efficiënte methode om dit te doen werd voorgesteld in [Wit10]: grounden met grenzen. Deze methode wordt uitgewerkt in hoofdstuk 3.2.1. Daarna (hoofdstuk 3.2.2) bekijken we nog enkele mogelijke verbeteringen voor dit groundingsalgoritme.

### 3.2.1 Grounden met grenzen

Om de methode uit [Wit10] te kunnen uitwerken, hebben we eerst volgende definitie nodig:

**Definitie 3.2.** *Gegeven een formule  $\varphi[x_1, \dots, x_n]$ , zij dan  $\varphi^{ct}[x_1, \dots, x_{n-k}]$  en  $\varphi^{cf}[x_1, \dots, x_{n-l}]$  twee formules over  $\sigma$ , waarvoor geldt dat*

$$\begin{aligned} \mathcal{T} \models (\forall(x_1, \dots, x_n)(\varphi^{ct}[x_1, \dots, x_{n-k}] \Rightarrow \varphi[x_1, \dots, x_n])) \text{ en,} \\ \mathcal{T} \models (\forall(x_1, \dots, x_n)(\varphi^{cf}[x_1, \dots, x_{n-l}] \Rightarrow \neg\varphi[x_1, \dots, x_n])). \end{aligned}$$

---

<sup>5</sup>Met diepte van een kwantor bedoelen we het aantal kwantoren rond de formule waarin deze kwantor voorkomt, oftewel hoe diep deze kwantor genest is.

We noemen formules  $\varphi^{ct}$ ,  $\varphi^{cf}$  die hieraan voldoen een **certainly-true** respectievelijk een **certainly-false grens** voor  $\varphi$  over  $\mathcal{T}$ . Een afbeelding  $\mathcal{C}$  die alle subformules  $\psi$  van formules in  $\mathcal{T}$  afbeeldt op een koppel  $(\psi^{ct}, \psi^{cf})$  van certainly-grenzen noemen we een **certainly-afbeelding (c-afbeelding)** van  $\mathcal{T}$  over  $\sigma$ .

We introduceren dus voor elke eerste-ordeformule  $\varphi$  een paar nieuwe formules  $\varphi^{ct}$  en  $\varphi^{cf}$ , waar we de interpretatie van kennen, en zodat  $\mathcal{T} \models \varphi^{ct}$  impliceert dat  $\mathcal{T} \models \varphi$  en zodat  $\mathcal{T} \models \varphi^{cf}$  impliceert dat  $\mathcal{T} \models \neg\varphi$ . Hiermee kunnen we algoritme 3.1 aanpassen, zodat er enkel nog een grounding wordt uitgevoerd voor formules waarover we nog geen zekerheid hebben (zie algoritme 3.2). Merk op dat **f** voor elke formule zowel een certainly-true als een certainly-false grens is.

We bekijken algoritme 3.2 eens van dichterbij. Wat we in feite doen, is elke grounding van een formule  $\varphi$  vervangen door een grounding van  $(\varphi \wedge \neg\varphi^{cf}) \vee \varphi^{ct}$ . Hoewel deze twee formules equivalent zijn in elk model van  $\mathcal{T}$ , zijn ze niet logisch equivalent, zoals duidelijk te zien is in dit voorbeeld:

**Voorbeeld 3.3.** Neem een theorie  $\mathcal{T}$  die bestaat uit een zin  $\varphi = \forall xP(x)$ . Stel dan dat  $\varphi^{ct} = \mathbf{t}$ , en  $\varphi^{cf} = \mathbf{f}$ . Merk op dat dit goede certainly-grenzen zijn voor  $\varphi$  aangezien er geldt dat

$$\begin{aligned}\mathcal{T} &\models \varphi \\ &\models \mathbf{t} \Rightarrow \varphi \\ &\models \varphi^{ct} \Rightarrow \varphi\end{aligned}$$

$$\begin{aligned}\mathcal{T} &\models \mathbf{t} \\ &\models \mathbf{f} \Rightarrow \varphi \\ &\models \varphi^{cf} \Rightarrow \varphi\end{aligned}$$

Maar anderzijds geldt duidelijk dat  $\varphi$  niet logisch equivalent is met **t**, maar dat  $(\varphi \wedge \neg\varphi^{cf}) \vee \varphi^{ct}$  wel heel duidelijk een tautologie is. Dus stel dat we algoritme 3.2 zouden gebruiken met voor  $\mathcal{T}$  met deze grenzen, dan zou op regel 27 de test  $I \not\models \psi_d^{ct}$  altijd falen, waardoor **t** als grounding van  $\varphi$  zou terugkomen, wat natuurlijk niet correct is.

Het probleem is dat de getransformeerde theorie modellen heeft waarin de voorwaarden van de grenzen  $(\varphi^{ct} \Rightarrow \varphi$  en  $\varphi^{cf} \Rightarrow \neg\varphi)$  niet meer voldaan zijn. Als we dus willen zorgen dat onze gereduceerde grounding nog steeds correct is, moeten we waarborgen dat in elk model voor onze theorie de voorwaarden voor de grenzen bewaard blijven. We kunnen dus met andere woorden een formule  $\varphi$  in een theorie  $\mathcal{T}$  niet vervangen door  $(\varphi \wedge \neg\varphi^{cf}) \vee \varphi^{ct}$ , maar wel door de conjunctie van volgende drie formules:

$$\begin{aligned}(\varphi \wedge \neg\varphi^{cf}) \vee \varphi^{ct} \\ \varphi^{ct} \Rightarrow \varphi \\ \varphi^{cf} \Rightarrow \neg\varphi\end{aligned}$$

Dus als we een theorie willen grounden, en we willen algoritme 3.2 gebruiken, dan moeten we voor elke formule  $\varphi$  die we grounden, ook  $\varphi^{ct} \Rightarrow \varphi$  en  $\varphi^{cf} \Rightarrow \neg\varphi$  grounden. Hoe



**Algoritme 3.2:** Grounding met grenzen van een zin  $\varphi$ 

**input** : Een zin  $\varphi \in \mathcal{L}_{\mathbf{FO}, \Sigma}$ , een interpretatie  $I$  voor  $\sigma \subset \Sigma$  en een  $c$ -afbeelding  $\mathcal{C}$

**output:** Een propositionele zin  $\varphi'$

```

1  switch  $\varphi$  do
2    case  $\varphi$  is (negatie van) atomische formule
3      | return  $\varphi$ 
4    case  $\varphi = \varphi_1 \vee \dots \vee \varphi_n$ 
5      | if  $I \models \varphi_i^{ct}$  voor een  $i \in [1..n]$  then return t
6      | else
7        |    $G := \emptyset$ ;
8        |   for  $i = 1 \dots n$  do
9          |     if  $I \not\models \varphi_i^{cf}$  then  $G := G \cup \text{ground}(\varphi_i, I, \mathcal{C})$ ;
10         |   end
11         |   if  $G = \emptyset$  then return f else return  $G_1 \vee \dots \vee G_n$ 
12         |   // Waarbij  $G = \{G_1, \dots, G_n\}$ 
13       | end
14    case  $\varphi = \varphi_1 \wedge \dots \wedge \varphi_n$ 
15      | if  $I \models \varphi_i^{cf}$  voor een  $i \in [1..n]$  then return f
16      | else
17        |    $G := \emptyset$ ;
18        |   for  $i = 1 \dots n$  do
19          |     if  $I \not\models \varphi_i^{ct}$  then  $G := G \cup \text{ground}(\varphi_i, I, \mathcal{C})$ 
20          |   end
21          |   if  $G = \emptyset$  then return t else return  $G_1 \wedge \dots \wedge G_n$ 
22          |   // Waarbij  $G = \{G_1, \dots, G_n\}$ 
23        | end
24    case  $\varphi = \forall x \psi[x]$ 
25      |  $G := \emptyset$ ;
26      | for  $d \in D_x$  do
27        |   if  $I \not\models \psi[d]^{ct}$  then
28          |     if  $I \models \psi[d]^{cf}$  then return f else  $G := G \cup \text{ground}(\psi[d], I, \mathcal{C})$ 
29        |   end
30      | end
31      | if  $G = \emptyset$  then return t else return  $G_1 \wedge \dots \wedge G_n$ 
32      | // Waarbij  $G = \{G_1, \dots, G_n\}$ 
33    case  $\varphi = \exists x \psi[x]$ 
34      |  $G := \emptyset$ ;
35      | for  $d \in D_x$  do
36        |   if  $I \not\models \psi[d]^{cf}$  then
37          |     if  $I \models \psi[d]^{ct}$  then return t else  $G := G \cup \text{ground}(\psi[d], I, \mathcal{C})$ 
38        |   end
39      | end
40      | if  $G = \emptyset$  then return f else return  $G_1 \vee \dots \vee G_n$ 
41      | // Waarbij  $G = \{G_1, \dots, G_n\}$ 
42
43 end

```

beter (scherper) we onze certainly-true en certainly-false grenzen kunnen maken, hoe makkelijker en kleiner de grounding van  $(\varphi \wedge \neg \varphi^{cf}) \vee \varphi^{ct}$  wordt<sup>6</sup>, aangezien er steeds vlugger kan worden “ingegrepen” om overbodige groundings van deelformules te vermijden. Stel dat we ter illustratie de formule  $\forall x(P(x)) \wedge \forall x, y, z(Q(y) \vee R(x, y, z))$  grounden, met  $D_x = D_y = D_z = [1 \dots 1000] \subset \mathbb{N}$  maar we weten dat  $\forall x(P(x))$  certainly-false grens  $\exists x Q(x)$  heeft, en dat  $Q(1)$  geldt in  $I$ . Dan wordt deze grounding al na de eerste stap stopgezet, en wordt er **f** teruggegeven. Anderzijds, geldt wel dat we de formules  $\varphi^{ct} \Rightarrow \varphi$  en  $\varphi^{cf} \Rightarrow \neg \varphi$  enkel moeten grounden als we de grenzen  $\varphi^{ct}$  en  $\varphi^{cf}$  gebruiken. Dit volgt uit het feit dat als  $I \not\models \varphi^{ct}$ , dat dan

$$\begin{aligned} \varphi^{ct} \Rightarrow \varphi &\equiv \neg \varphi^{ct} \vee \varphi \\ &\equiv \neg \mathbf{f} \vee \varphi \\ &\equiv \mathbf{t} \end{aligned}$$

in de interpretatie  $I$ . Dus deze grounding moeten we enkel uitvoeren als  $I \models \varphi^{ct}$ , wat dus vaker het geval is wanneer de grenzen beter worden. Een analoog resultaat geldt voor  $\varphi^{cf} \Rightarrow \neg \varphi$ . Dit betekent dus dat hoe scherper we de grenzen maken, hoe groter de grounding van  $\varphi^{ct} \Rightarrow \varphi$  en  $\varphi^{cf} \Rightarrow \neg \varphi$  zal worden<sup>6</sup>. Bij het opstellen van de grenzen, moeten we ook zorgen dat ze niet te complex worden, aangezien het uitvoeren van een test van de vorm  $I \models \varphi^{ct}$  in principe een *modelchecking*-probleem is (= modelexpansie met  $\Sigma = \sigma$ ), en modelchecking een PSPACE-compleet probleem is [Sto74].

We moeten dus goed onderzoeken hoe we deze grenzen dan het beste opstellen, want zoals het hierboven geformuleerd wordt, lijkt het alsof het gebruik van deze grenzen eigenlijk niets vereenvoudigt aan de grounding. Het algoritme zegt namelijk dat we telkens we de grens  $\varphi^{ct}$  zouden kunnen gebruiken, we in de plaats  $\varphi^{ct} \Rightarrow \varphi$  moeten grounden. Als we dus in de praktijk werkelijk het grounden van  $\varphi^{ct} \Rightarrow \varphi$  moeten toevoegen, is het algoritme nogal zinloos. Stel nu echter dat we onze grenzen zo kunnen opstellen, zodat we voor de test of  $I \models \varphi^{ct} \Rightarrow \varphi$ , deze formule niet volledig moeten grounden. In [WMD10] wordt het volgende voorgesteld: Stel dat we zorgen dat we onze grenzen zo opstellen dat voor elke formule  $\varphi$  geldt dat

$$\begin{aligned} (\bigwedge \psi_i^{ct} \Rightarrow \psi_i) &\Rightarrow \varphi^{ct} \Rightarrow \varphi \\ (\bigwedge \psi_i^{cf} \Rightarrow \neg \psi_i) &\Rightarrow \varphi^{cf} \Rightarrow \neg \varphi \end{aligned}$$

We bedoelen hier met  $\{\psi_i\}$  de verzameling atomische formules in  $\varphi$ . Dan geldt dat het voldoende is enkel de formules  $\tau_i^{ct} \Rightarrow \tau_i$  en  $\neg \tau_i^{ct} \Rightarrow \tau_i$  te grounden, waarbij we met  $\{\tau_i\}$  de verzameling atomische formules in  $\mathcal{T}$  bedoelen. Dit is natuurlijk niet veel werk, aangezien de certainly-grenzen formules over  $\sigma$  zijn, en er in elk van die te grounden formules maximaal één atomische formule staat die we echt moeten grounden.

### Opstellen van grenzen

In het IDP-systeem wordt een algoritme gebruikt, zoals voorgesteld in [WMD10], om de grenzen op te stellen. We beschrijven kort hoe dit algoritme in zijn werk gaat. Het

<sup>6</sup>Voor een formeel bewijs hiervan, zie [WMD10].

algoritme is bedoeld om recursief gebruikt te worden, om bij elke herhaling de grenzen bij te stellen. Op deze manier kan het zonder problemen na een bepaalde tijd stopgezet worden, zonder dat er teveel werk verloren gaat. Initieel geven we elke formule in  $\mathcal{T}$  certainly-true en certainly-false grens **f**. Elke herhaling van het algoritme wordt er één van de volgende verbeteringsmethoden toegepast waarna  $\varphi^{ct}$  vervangen wordt door  $\varphi_{new}^{ct} \vee \varphi^{ct}$ , en  $\varphi^{cf}$  vervangen wordt door  $\varphi_{new}^{cf} \vee \varphi^{cf}$ . In [WMD10] worden er zes verbeteringsmethoden voorgesteld, we bekijken deze methoden hier kort:

- **Input verbetering:** Als  $\varphi$  een formule over  $\sigma$  is, dan is het logisch dat  $\varphi^{ct} = \varphi$  en  $\varphi^{cf} = \neg\varphi$ .
- **Axioma verbetering:** Als  $\varphi$  een zin is in  $\mathcal{T}$ , dan geldt dat  $\varphi^{ct} = \mathbf{t}$  en  $\varphi^{cf} = \mathbf{f}$ , dat hebben we hierboven in een voorbeeld al aangetoond.
- **Bottom-Up verbetering:** Als bijvoorbeeld  $\varphi = \varphi_1 \vee \varphi_2$ , dan is  $\varphi' = \varphi_1^{ct} \vee \varphi_2^{ct}$  een certainly-true grens voor  $\varphi$ , dus noem  $\varphi^{ct} = \varphi'$ . Voor alle gevallen, zie [WMD10].
- **Top-Down verbetering:** Neem een formule  $\varphi[\bar{x}, \bar{y}, \bar{z}] = \varphi_1[\bar{x}, \bar{z}] \wedge \varphi_2[\bar{x}, \bar{y}]$ , dan geldt dat  $\varphi_1^{ct} = \exists \bar{y}(\varphi^{ct})$  en  $\varphi_2^{ct} = \exists \bar{z}(\varphi^{ct})$ . Voor alle gevallen, zie [WMD10].
- **Functionele verbetering:** Deze verbetering kunnen we toepassen als  $\varphi[\bar{x}, y] = F(\bar{x}) = y$  voor een bepaalde functie  $F$ . We kunnen dan speciale eigenschappen van functies gebruiken voor grenzen op te stellen. Functies hebben onder meer de eigenschap dat voor alle  $y$  er precies één  $\bar{x}$  bestaat zodat  $F(\bar{x}) = y$ . Dit wil dus zeggen dat als  $F(\bar{x}) = y$  zeker waar is, dat dan voor alle  $y' \neq y$  de formule  $F(\bar{x}) = y'$  zeker niet waar is. En omgekeerd dat als voor alle  $y' \neq y$  geldt dat  $F(\bar{x}) \neq y'$ , dan moet  $F(\bar{x}) = y$  zeker waar zijn. Formeel geeft dit dus:

$$\begin{aligned}\varphi_{new}^{ct} &:= \forall y'(y \neq y' \implies \varphi^{cf}[y/y']) \\ \varphi_{new}^{cf} &:= \exists y'(\varphi^{ct}[y/y'] \wedge y \neq y')\end{aligned}$$

- **Verbetering door het opsporen van dubbels:** Stel dat we twee formules  $\varphi[x_1, \dots, x_n]$  en  $\psi[y_1, \dots, y_n]$  hebben waarvoor we weten dat  $\varphi[x_1/z, \dots, x_n/z]$  en  $\psi[y_1/z, \dots, y_n/z]$  equivalent zijn. Noem dan

$$E(\bar{x}, \bar{y}) = \bigwedge_{i=1..n} x_i = y_i$$

Dan kunnen we de certainly grenzen van de ene verbeteren door informatie die we al bij de andere gevonden hebben te gebruiken. Meer specifiek kunnen we nieuwe grenzen voor  $\varphi$  als volgt opstellen:

$$\begin{aligned}\varphi_{new}^{ct} &:= \exists \bar{y}(\psi^{ct} \wedge E(\bar{x}, \bar{y})) \\ \varphi_{new}^{cf} &:= \exists \bar{y}(\psi^{cf} \wedge E(\bar{x}, \bar{y}))\end{aligned}$$

Merk op dat we hiervoor equivalente formules moeten kunnen herkennen, wat voor eerste-ordelogica geen sinecure is. In de praktijk gaan we hiervoor FOBDDs gebruiken (hoofdstuk 2). Het is dus duidelijk in ons voordeel dat we semantisch equivalente FOBDDs zo goed mogelijk kunnen herkennen. We gaan hier dieper op in in hoofdstuk 3.2.3.

De volgorde waarin deze methoden worden toegepast, en op welke formules ze worden toegepast, wordt bepaald met allerlei heuristieken waar we hier niet dieper op ingaan. In [WMD10] worden deze methoden uitvoerig getest op allerlei voorbeelden, en men komt daar tot volgende opmerkingen:

- We moeten een goede manier vinden om gevonden grenzen te vereenvoudigen. Als we bijvoorbeeld een grens  $\varphi^{ct}$  willen opstellen voor een formule  $\varphi$ , die een samenstelling is van  $\varphi_1$  en  $\varphi_2$ , en we gebruiken Bottom-Up verbetering, dan wordt  $\varphi^{ct}$  een samenstelling van  $\varphi_1^{ct}$  en  $\varphi_2^{ct}$ . We zien dus dat als we grenzen nooit vereenvoudigen, dat ze dan enkel groter en complexer kunnen worden, met als gevolg dat tests van de vorm  $I \models \varphi^{ct}$  erg duur zullen worden.
- We moeten een goede manier vinden om te weten te komen of twee grenzen gelijk kunnen zijn (merk op dat dit altijd met een benaderende methode zal moeten gebeuren, aangezien dit probleem onbeslisbaar is), zodat we kunnen beslissen of de methode een fixpoint<sup>7</sup> bereikt.
- We moeten een goed stopcriterium hebben in het geval dat een fixpoint niet, of niet snel genoeg bereikt wordt.

Om aan deze eisen tegemoet te komen, kunnen we de interessante normaalvorm gebruiken die we hebben gedefinieerd in hoofdstuk 2, namelijk FOBDDs. We hebben in dat hoofdstuk al opgemerkt dat FOBDDs de interessante eigenschap hebben dat FOBDDs van twee equivalente formules vaak syntactisch gelijk zijn. Een tweede voordeel is dat het een heel handige normaalvorm is voor het vereenvoudigen van de formules (waarop we verder ingaan in hoofdstuk 3.2.3). Nog een ander voordeel van FOBDDs is dat logische bewerkingen erop goedkoop zijn. Zoals we in hoofdstuk 2.2.2 gezien hebben kunnen we van twee FOBDDs bijvoorbeeld gemakkelijk de conjunctie of de disjunctie nemen. De laatste voorname reden waarom we FOBDDs gebruiken is dat er elegante algoritmen bestaan om query's op te lossen met FOBDDs. Dan kunnen we in plaats van in algoritme 3.2 telkens over alle domein elementen te lopen en testen of ervoor geldt dat  $I \models \psi[d]^{ct}$ , lopen over alle oplossingen van de query  $\{d \in D_x \mid \varphi[x/d]\}$ . In [Wit10] wordt een query-algoritme met FOBDDs uitgewerkt, we gaan er hier niet dieper op in.

### 3.2.2 Equivalente subformules herkennen

Zoals we verschillende keren eerder vermeld hebben, is een interessante eigenschap van FOBDDs dat twee FOBDDs voor twee equivalente formules vaak syntactisch gelijk zijn. Deze eigenschap willen we zoveel mogelijk uitbuiten in ons groundingsalgoritme. We maken al gebruik van deze eigenschap bij het opstellen van onze grenzen (Verbetering door het opsporen van dubbels p. 36), maar stel nu dat we meerdere keren een equivalente formule moeten grounden. Dan willen we dat zo frequent mogelijk kunnen gebruiken,

---

<sup>7</sup>Een fixpoint is een moment in de berekening waarop elke volgende verbeteringsmethode geen verbetering meer brengt, en de grenzen dus optimaal zijn.

om de grounding maar éénmalig te moeten uitvoeren. Stel bijvoorbeeld dat we over een bepaald vocabularium  $\Sigma$  een theorie  $\mathcal{T}$  met volgende formules moeten grounden:

$$\mathcal{T} = \left\{ \begin{array}{l} \forall \bar{x} (\varphi[\bar{x}] \wedge \psi_0 \wedge \psi_1 \vee \psi_2), \\ \forall \bar{x} (\tau_0 \vee \varphi[\bar{x}] \vee \tau_1), \\ \forall \bar{x} (\varphi[\bar{x}]) \end{array} \right\}$$

Waarbij  $\varphi$  een erg grote formule is. Dan is het zinvol om een nieuw Tseitin predicaat  $P$  in te voeren, en  $\mathcal{T}$  te vervangen door een nieuwe theorie  $\mathcal{T}'$  (over een nieuw vocabularium dat  $P$  bevat)<sup>8</sup>:

$$\mathcal{T}' = \left\{ \begin{array}{l} \forall \bar{x} (P[\bar{x}] \wedge \psi_0 \wedge \psi_1 \vee \psi_2), \\ \forall \bar{x} (\tau_0 \vee P[\bar{x}] \vee \tau_1), \\ \forall \bar{x} P[\bar{x}], \\ \forall \bar{x} (P[\bar{x}] \Leftrightarrow \varphi[\bar{x}]) \end{array} \right\}$$

Nu geldt dat er een canonieke bijectie bestaat tussen de modellen van  $\mathcal{T}$  en de modellen van  $\mathcal{T}'$ . Hiermee bedoelen we dat elk model van  $\mathcal{T}'$  kan beperkt worden tot een model in  $\mathcal{T}$  en dat elk model van  $\mathcal{T}$  kan worden uitgebreid tot een model van  $\mathcal{T}'$ . We mogen dus verder  $\mathcal{T}'$  grounden in plaats van  $\mathcal{T}$ . Het voordeel van het grounden van  $\mathcal{T}'$  is dat we de formule  $\varphi$  maar één keer grounden, en deze grounding telkens kunnen gebruiken als we  $P$  tegenkomen in een andere formule.

Om dit te kunnen doen moeten we eerst en vooral herkennen dat die formule  $\varphi$  zo vaak voorkomt. We willen bovendien niet enkel weten of die formule  $\varphi$  op zich vaak voorkomt, maar ook of er equivalente formules  $\varphi'$  zijn die ook voorkomen in de theorie. Hiervoor kunnen we de eigenschap van FOBDDs gebruiken, dat equivalente formules, vaak dezelfde FOBDD hebben, of dat we equivalente subformules kunnen herkennen door het detecteren van gelijke deelbomen.

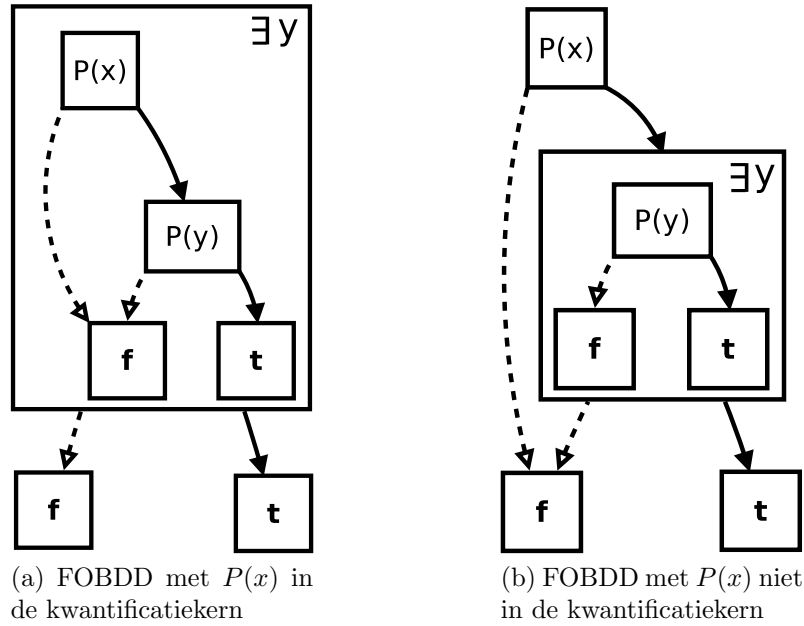
### 3.2.3 Vereenvoudigen van FOBDDs

De vorige hoofdstukken is er al meermaals aangehaald dat we graag gebruik maken van het feit dat de FOBDDs van twee equivalente formules vaak syntactisch gelijk zijn. Voor propositionele formules (BDDs) weten we zelfs dat dit altijd het geval is. Voor FOBDDs geldt dit resultaat echter niet, we kunnen semantische equivalentie tussen FOBDDs niet controleren omdat dit onbeslisbaar is<sup>9</sup>. We kunnen dus enkel syntactische gelijkheid controleren, maar dit kunnen we wel in constante tijd, aangezien twee syntactisch gelijke FOBDDs op dezelfde plaats in het geheugen staan opgeslagen.

In dit hoofdstuk gaan we proberen om de FOBDDs die we hebben verder te vereenvoudigen en te standaardiseren, zodat we in nog meer gevallen equivalentie tussen FOBDDs kunnen ontdekken. Buiten het ontdekken van die equivalenties, hebben we nog redenen waarom

<sup>8</sup>We gebruiken de notatie  $\bar{x}$  als afkorting voor  $x_1, x_2, \dots, x_n$

<sup>9</sup>Merk op dat dit niet enkel wil zeggen dat FOBDDs geen canonieke vorm voor eerste-orde logica kunnen zijn, maar zelfs dat er geen canonieke vorm voor eerste-orde logica kan bestaan.

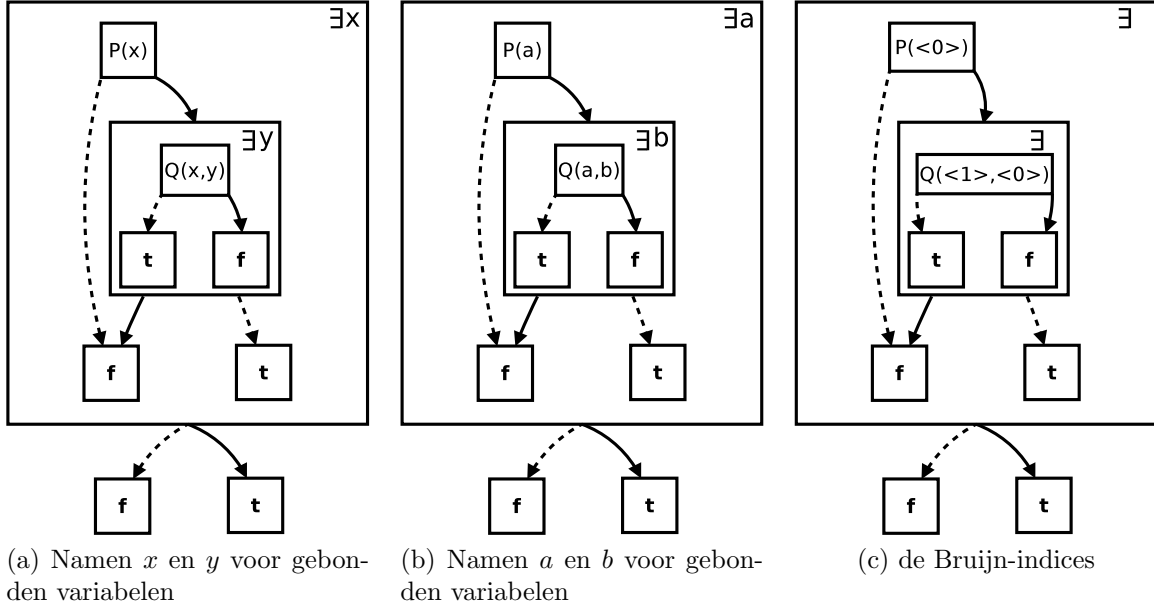


Figuur 3.1: Valse afhankelijkheid in een FOBDD

we FOBDDs zo eenvoudig mogelijk willen maken, namelijk de opmerkingen op het einde van hoofdstuk 3.2.1. Ten eerste geldt dat als de FOBDDs die de query's voorstellen bij het grounden met grenzen eenvoudig zijn, dat we dan het queryen sneller kunnen uitvoeren. Een ander voordeel van eenvoudige FOBDDs, is dat we bij het opstellen van grenzen vaker en sneller kunnen herkennen of de methode die de grenzen bepaalt een fixpoint bereikt.

Dit vereenvoudigen van FOBDDs is onder meer onderzocht in [Gou95], waar men tot een algoritme *simplify* gekomen is. Dit algoritme heeft als hoofddoel het beperken van valse afhankelijkheid in FOBDDs. Bekijk om dit te verduidelijken figuur 3.1, daar zien we twee FOBDDs van een equivalente formule, die niet syntactisch gelijk zijn. In figuur 3.1a zien we een kern met kwantificatie  $\exists y$ , waarin het predicaat  $P(x)$  zit. Dit bedoelen we met valse afhankelijkheid, hoewel  $P(x)$  in die kern voorkomt, zou dit eigenlijk niet moeten, want  $P(x)$  is onafhankelijk van  $y$ . We willen in een kwantificatiekern enkel predicaten hebben die afhankelijk zijn van de gekwantificeerde variabele. We gaan hier niet in op de concrete implementatie van het algoritme, en verwijzen daarvoor naar [Gou95].

Een volgend probleem met FOBDDs is de naamgeving voor gebonden variabelen. In figuur 3.2a, 3.2b zien we weer twee FOBDDs die equivalent zijn, maar niet syntactisch gelijk. Hier is het probleem dat we in figuur 3.2a de gekwantificeerde variabele een andere naam heeft dan in figuur 3.2b. Om dit op te lossen moeten we dus een methode vinden om die (redundante) namen te kunnen vervangen door iets wat niet van naamgeving afhankelijk is. Een gekende methode hiervoor is het gebruik van de Bruijn-indices ([dB72]), die de naam van de variabele vervangt door een verwijzing naar de kwantor. Dit doen we concreet door de naam te vervangen door een index die aangeeft hoeveel kwantoren we in een formule (of FOBDD) naar buiten moeten om bij de bijhorende kwantor te geraken. In figuur 3.2c zien we opnieuw dezelfde FOBDD, waarbij de namen van de variabelen



Figuur 3.2: Het gebruik van de Buijn-indices in een FOBDD

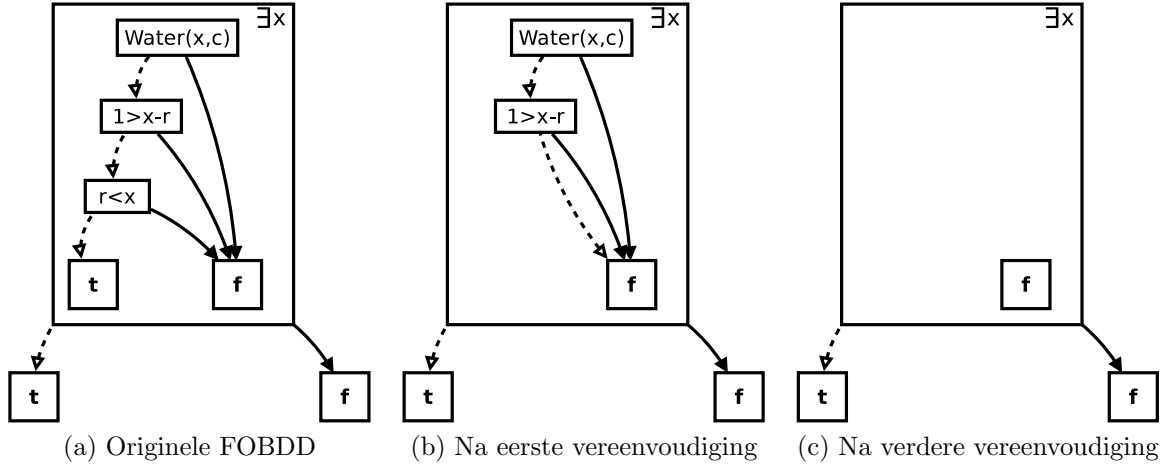
vervangen zijn door de Buijn-indices. De kwantor voor  $x$  was de buitenste, dus binnen de kwantificatiekern van  $y$  moeten we één kwantor “overslaan” om die bij  $x$  horende te vinden, vandaar krijgt  $x$  daar de Buijn-index 1. Om de kwantor voor  $y$  van hieruit te vinden moeten we er geen overslaan, want dit is de binnenste kwantor,  $y$  krijgt dus index 0.

Buiten deze vereenvoudigingen werden er in [Wit10] nog enkele extra kleine vereenvoudigingen voorgesteld:

- Een FOBDD van de vorm  $x = x \rightarrow \varphi_1, \varphi_0$  vervangen we door  $\varphi_1$ .
- Een FOBDD van de vorm  $\exists x(F(\bar{y}) = x) \rightarrow \varphi_1, \varphi_0$  vervangen we door  $\varphi_1$  als  $F$  een totale functie is. Het teruggeef-type van  $F$  is hetzelfde als het type van  $x$ .
- Bij het gebruik van aritmetiek in FOBDDs zetten we alle aritmetische formules altijd in een normaalvorm zodat onder andere de FOBDD voor  $\exists x(\exists y(x = y))$  en de FOBDD voor  $\exists x(\exists y(x - y = 0))$  syntactisch gelijk zijn.

### 3.2.4 Aritmetische redundantie

Alle vereenvoudigingen die hierboven besproken werden, zijn allemaal vereenvoudigingen die op dit moment in het IDP-systeem geïmplementeerd zijn, en die werken. In de context van deze thesis heb ik getracht om dit nog verder uit te breiden, meer specifiek voor FOBDDs die aritmetiek bevatten. Neem nu bijvoorbeeld twee formules  $\varphi = (x > 5) \wedge (x > 3)$  en  $\psi = (x > 5)$ . Het is duidelijk dat ze equivalent zijn, maar als we van beide een



Figuur 3.3: Aritmetische redundante in een FOBDD voor  $\{r, c \mid \forall x(Water(x, c) \vee 1 > x - r \vee r < x)\}$

FOBDD maken, gaan deze niet syntactisch gelijk zijn. De FOBDD voor  $\varphi$  zal namelijk een (weliswaar overbodige) knoop met formule  $x > 3$  bevatten, die de FOBDD voor  $\psi$  niet bevat.

Bij het gebruik van aritmetiek kunnen er dus allerlei redundancies voorkomen in FOBDDs. Bekijk als groter voorbeeld figuur 3.3. Dit is een FOBDD met betrekking tot de grenzen van het battleship probleem, en komt overeen met de query:

$$\{r, c \mid \forall x(Water(x, c) \vee 1 > x - r \vee r < x)\}$$

Op het moment dat er getest wordt of  $r < x$ , hebben we eigenlijk al wat informatie over  $r$  en  $x$  die we kunnen gebruiken. Het gegeven dat deze knoop bereikt wordt, impliceert onder meer dat  $\neg(1 > x - r)$ , oftewel dat  $1 + r \leq x$ . Hieruit volgt dan onmiddellijk dat  $r < x$  moet gelden, en dat de **t**-knoop in de kwantificatiekern nooit bereikt kan worden. We kunnen dus de knoop met  $r < x$  weghalen, en de ouder verbinden met de **f**-knoop. Dit zien we in figuur 3.3b. We merken nu dat zowel de truetak als de falsetak van de knoop met formule  $1 > x - r$  naar de **f**-knoop verwijzen. Deze knoop kunnen we dus ook weghalen, en dan verbinden we de falsetak van  $Water(x, c)$  met de **f**-knoop. Als we dit verderzetten merken we dat de FOBDD in figuur 3.3a equivalent is met die in figuur 3.3c, welke verder vereenvoudigt tot **t**. De formule  $\forall x(Water(x, c) \vee 1 > x - r \vee r < x)$  was dus een tautologie.

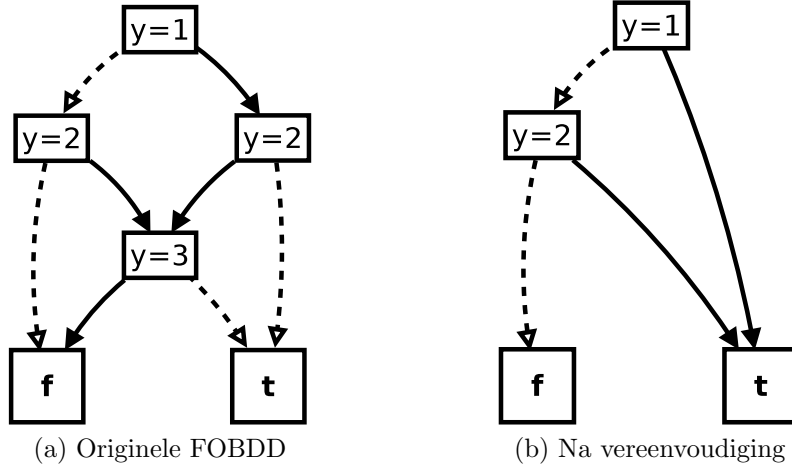
Een andere reden dat we deze redundantie willen wegwerken zien we in figuur 3.4. Deze FOBDD stelt volgende query voor:

$$\{y[\mathbb{N}] \mid (y = 1 \vee y = 2) \wedge (y \neq 2 \vee y \neq 3)\}$$

De oplossing voor deze query is duidelijk, namelijk  $y \in \{1, 2\}$ . Als we de query echter oplossen via het query-algoritme dat gebruikt wordt in IDP<sup>10</sup>, dan duurt dit oneindig

<sup>10</sup>Dit algoritme is een optimalisatie van het query-algoritme in [WMD10].





Figuur 3.4: Aritmetische redundante in een FOBDD voor  $\{y[\mathbb{N}] \mid (y = 1 \vee y = 2) \wedge (y \neq 2 \vee y \neq 3)\}$

lang. Het algoritme gaat namelijk voor alle natuurlijke getallen testen of ze 1 of 2 zijn, en niet 2 of 3. Als we echter vereenvoudigingen op de FOBDD doorvoeren, die ons de FOBDD in figuur 3.4b geven, dan is dit niet meer het geval. Het query-algoritme ontdekt dan dat de true tak van  $x = 3$  de knoop  $t$  is, en dat geldt dat

$$\begin{aligned}
 \{y \mid (y = 1 \vee y = 2) \wedge (y \neq 2 \vee y \neq 3)\} &= \{y \mid (y = 1 \vee y = 2)\} \\
 &= \{y \mid y = 1\} \cup \{y \mid y = 2\} \\
 &= \{1, 2\}
 \end{aligned}$$

Het moet dan niet meer alle natuurlijke getallen overlopen, maar kan dit zelf afleiden uit de vorm van de FOBDD. Dit voorbeeld toont dus aan dat dit probleem niet enkel het IDP-systeem vertraagt, maar dat er zelfs problemen zijn die niet oplosbaar zijn, simpelweg door een beetje aritmetische redundantie in een FOBDD.

# Hoofdstuk 4

## Vereenvoudiging van aritmetische FOBDDs

### 4.1 Inleiding

In het vorige hoofdstuk hebben we reeds enkele voorbeelden gezien van FOBDDs die aritmetische formules bevatten, waar knopen in zitten die eigenlijk redundant zijn. In dit hoofdstuk beschrijven we een algoritme dat deze knopen kan opsporen en FOBDDs met deze informatie kan vereenvoudigen. Om de algoritmes en de bewijzen in dit hoofdstuk overzichtelijk te kunnen uitwerken bekijken we in dit hoofdstuk enkel FOBDDTs (eerste-ordeBDTs, zie hoofdstuk 2.2), die wel gereduceerd en geordend zijn, maar niet gedeeld. Merk op dat dit niets afdoet aan de stellingen en algoritmes, aangezien er een één op één verband is tussen de gereduceerde en geordende BDTs en de FOBDDs, zoals we al opgemerkt hebben in hoofdstuk 2.2. In hoofdstuk 4.7 bekijken we heel kort de aanpassingen die we aan het geconstrueerde algoritme moeten maken als we toch met FOBDDs werken.

Voor we aan een algoritme toekomen, definiëren we eerst enkele begrippen die we verder in dit hoofdstuk nodig hebben. We bekijken wat we precies bedoelen met een knoop die (aritmetisch) redundant is en we bekijken wat we daar precies aan zouden kunnen doen.

**Propositie 4.1.** *Zij  $B$  een FOBDDT, dan noemen we een rij knopen  $P = \{K_0, K_1, K_2, \dots, K_n\}$  een **pad**, als  $P$  voldoet aan:*

- $P$  is een pad in  $B$  bekeken als gerichte graaf<sup>1</sup>.
- $K_0$  is de wortel van  $B$ .
- Voor elke  $i \in [0, n-1]$  geldt dat  $K_{i+1}$  ofwel de wortel van de true- of van de falsetak van  $K_i$  is.

**Definitie 4.1.** *Zij  $P = \{K_0, K_1, K_2, \dots, K_n\}$  een pad in een FOBDDT  $B$ , definieer dan voor elke  $K_i$  ( $i \in [0, n-1]$ ) een formule  $\psi_i$  als volgt:*

---

<sup>1</sup>Voor een definitie van paden in de grafentheorie, zie [Die05].

- $\psi_i$  is de formule die  $K_i$  voorstelt als de volgende knoop op het pad de wortel van de truetak van  $K_i$  is.
- $\psi_i$  is de negatie van de formule die  $K_i$  voorstelt als de volgende knoop op het pad de wortel van de falsetak van  $K_i$  is

We noemen dan  $\psi_i$  de **padformule** voor  $K_i$  in  $P$ , en  $V = \{\psi_i \mid i \in [0, n-1]\}$  de **padformuleverzameling** voor  $P$ . Omdat in een FOBDT geldt dat er voor elke knoop  $K$  een uniek pad  $P$  naar  $K$  bestaat, spreken we ook soms over de padformuleverzameling van  $K$ .

Voor elk pad  $P$  met padformuleverzameling  $V = \{\psi_i \mid i \in [0, n-1]\}$  bestaat er dus een formule  $\psi = \bigwedge_{\psi_i \in V} \psi_i$  die dit pad voorstelt. Deze formules kunnen we gebruiken om het redundant zijn van een knoop te definiëren.

**Definitie 4.2.** We noemen een knoop  $K$  met formule  $\varphi$  in een FOBDT  $B$  **redundant** voor een theorie  $\mathcal{T}$  als er een pad  $P = \{K_0, K_1, K_2, \dots, K_n = K\}$  van de wortel  $K_0$  naar  $K$  bestaat, met padformuleverzameling  $V$ , waarvoor geldt dat:

$$\bigwedge_{\psi_i \in V} \psi_i \Rightarrow \varphi \quad \text{of} \quad \bigwedge_{\psi_i \in V} \psi_i \Rightarrow \neg\varphi$$

waar is in elk model van  $\mathcal{T}$ .

Als  $\varphi$  een aritmetische formule is, dan noemen we  $K$  **aritmetisch redundant** als

$$\bigwedge_{\psi_i \in W} \psi_i \Rightarrow \varphi \quad \text{of} \quad \bigwedge_{\psi_i \in W} \psi_i \Rightarrow \neg\varphi$$

waar is in elk model dat voldoet aan de theorie van de natuurlijke getallen<sup>2</sup>. Met  $W \subset V$  bedoelen we de verzameling aritmetische formules in  $V$ .

Stel nu dat we een FOBDT  $B$  hebben, waarvoor we willen controleren of een bepaalde knoop  $K$  op  $B$  aritmetisch redundant is. We moeten daarvoor eerst en vooral de verzameling  $W$  met aritmetische formules in de padformuleverzameling van  $K$  kennen. Eens we van elke knoop de padformuleverzameling kennen, moeten we de eigenlijke test of

$$\bigwedge_{\psi_i \in W} \psi_i \Rightarrow \varphi \quad \text{of} \quad \bigwedge_{\psi_i \in W} \psi_i \Rightarrow \neg\varphi$$

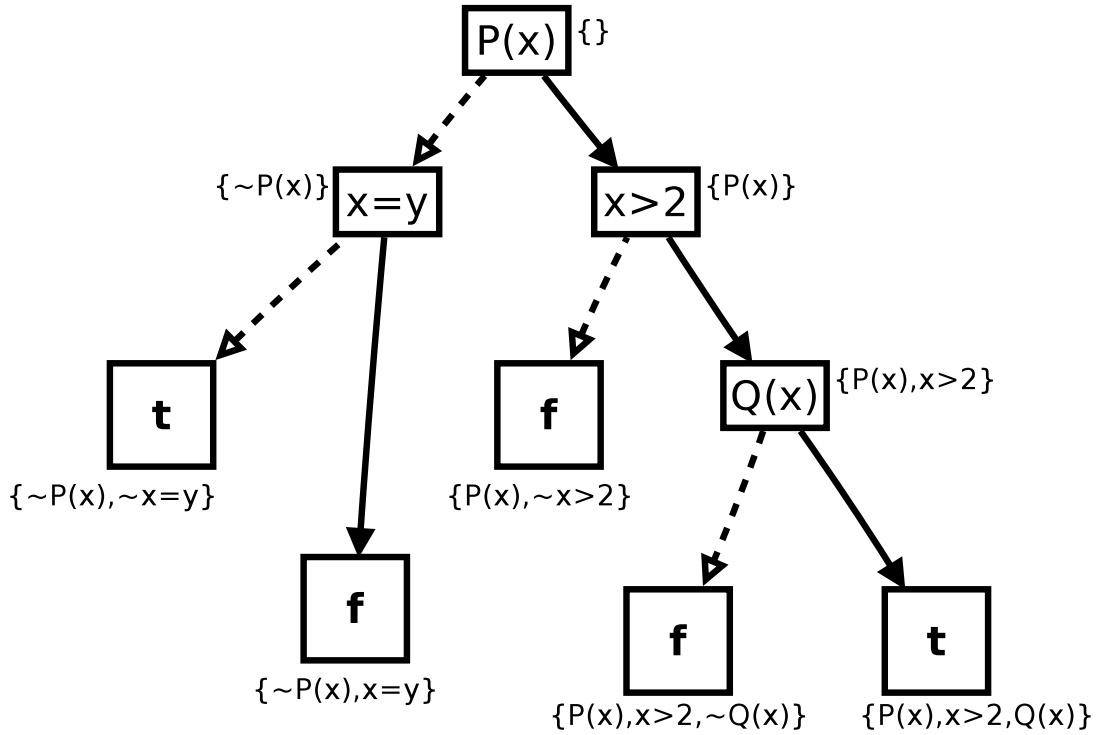
een tautologie is, zelf nog uitvoeren. Dit is niet vanzelfsprekend, aangezien elke theorie die sterk genoeg is om de natuurlijke getallen te kunnen beschrijven, (bv. Peano) onbeslisbaar is ([God31]). Dit wil dus zeggen dat er formules bestaan waar we niet in eindige tijd kunnen beslissen of ze een tautologie zijn. In hoofdstuk 4.3 onderzoeken we hoe we dit probleem (gedeeltelijk) kunnen omzeilen. Er wordt een algoritme om deze beslissing

<sup>2</sup>We bedoelen hiermee de theorie van de Peano Axioma's, voor meer hierover zie [CL01].

te maken geconstrueerd en er wordt besproken hoe we dit in de praktijk kunnen gebruiken. Op het moment dat we weten welke knopen redundant zijn, moeten we enkel de vereenvoudiging nog doorvoeren, dit wordt besproken in het uiteindelijke algoritme in hoofdstuk 4.4. Hierna bekijken we nog enkele uitbreidingen, en gaan we wat dieper in op de implementatie die we gedaan hebben in het IDP-systeem.

## 4.2 Een eerste versie van het algoritme

Zoals in de inleiding van dit hoofdstuk vermeld werd, is de eerste stap om te komen tot een werkend algoritme dat FOBDTs (en bijgevolg FOBDDs) kan vereenvoedigen, een efficiënte manier vinden om voor een knoop op een pad de padformuleverzameling te weten te komen. Om een beetje inzicht te krijgen in hoe zo een padformuleverzameling eruit ziet voor de knopen in een FOBDT, kijken we eerst naar figuur 4.1, waar bij elke knoop de padformuleverzameling van het pad naar die knoop gegeven is. Hier zien we goed dat voor elke knoop zijn padformule in alle padformuleverzamelingen van de paden die eindigen in een knoop in één van zijn takken liggen.



Figuur 4.1: De padformuleverzamelingen van alle knopen in een FOBDT

We gaan een algoritme construeren dat een hele FOBDT  $B$  recursief doorloopt, op elke knoop de verzameling van aritmetische formules  $W$  in de padformuleverzameling  $V$  kent, en  $B$  met die informatie vereenvoedigt. Om het algoritme te kunnen uitwerken stellen we een FOBDT (die geen triviale FOBDT is) als volgt voor: Een FOBDT  $B$  bestaat uit een kern (wortel)  $R$  en twee FOBDTs  $B_0$  (de falsetak onder  $R$ ) en  $B_1$  (de truetak onder  $R$ ).

Elke keer dat een FOBDT doorlopen wordt eerst getest of zijn kern aritmetisch redundant is. Als de kern niet redundant is, maar toch een aritmetische formule bevat, wordt de formule in de kern aan  $W$  toegevoegd en wordt de truetak bezocht. Hierna vervangen we de formule van de kern op de lijst door zijn negatie en bezoeken we de falsetak. Als we terugkomen van de falsetak, verwijderen we de formule uit de verzameling en geven we de vereenvoudigde versie van de FOBDT terug. Als de kern wel aritmetisch redundant is, zijn er twee mogelijkheden. Als

$$\bigwedge_{\psi_i \in W} \psi_i \Rightarrow \varphi$$

dan geldt dus dat altijd de truetak genomen zal worden, en mogen we deze knoop en zijn falsetak verwijderen. We vervangen dan deze FOBDT door zijn truetak. Als anderszijds

$$\bigwedge_{\psi_i \in W} \psi_i \Rightarrow \neg \varphi$$

dan vervangen we om analoge redenen deze FOBDT door zijn falsetak. In het speciale geval dat de kern van de FOBDT een kwantificatiekern zou zijn, vereenvoudigen we eerst de FOBDT in de kwantificatiekern, voor we overgaan tot het testen van de redundantie van die kern. Dit algoritme is in pseudocode te vinden in algoritme 4.1

### 4.3 Het beslissen van de redundantie

In algoritme 4.1 zien we onder meer op regel 9 de test  $\psi \Rightarrow \varphi$  staan, waarbij  $\psi$  en  $\varphi$  beide aritmetische formules zijn. Er is in de inleiding van dit hoofdstuk al kort aangehaald dat deze test heel wat meer inhoudt dan we zouden verwachten. Meer nog, in de praktijk is het zelfs niet altijd mogelijk dat we deze test in eindige tijd kunnen uitvoeren. Dit lijkt een erg grote tegenslag, aangezien dit wil zeggen dat we geen enkele methode kunnen vinden die we kunnen gebruiken om deze test uit te voeren. Dit betekent dus dat het onmogelijk is een algoritme te vinden dat alle aritmetisch redundante knopen kan opsporen. Gelukkig bestaan er wel methoden om een groot deel van deze knopen te vinden. De aanpak die in deze thesis gekozen werd, is om een variant op Peano aritmetiek te gebruiken, die dan wel niet meer alle aritmetische formules beschrijft, maar wel beslisbaar is. We hebben gekozen voor Presburger aritmetiek, de theorie van de gehele getallen zonder de vermenigvuldiging. In hoofdstuk 4.3.1 introduceren we deze theorie formeel, en we bewijzen dat we hem kunnen gebruiken voor ons algoritme (oftewel dat het beslisbaar is) in hoofdstuk 4.3.2. De praktische kant van de zaak bekijken we in hoofdstuk 4.3.3. Hier wordt een beslisser voorgesteld en een klein algoritme uitgewerkt dat deze beslissing neemt.

#### 4.3.1 Presburger aritmetiek

Als we kijken naar een theorie voor de natuurlijke getallen, dan denken we bijna spontaan aan de Peano aritmetiek. We kunnen de Peano aritmetiek formeel definiëren als een

---

**Algoritme 4.1:** simpBDT: aritmetische vereenvoudiging van FOBDTs
 

---

**Input:** Een FOBDT  $B$ 
**Interne variabelen:** Verzameling van aritmetische formules  $W$ 
**Output:** Aritmetisch vereenvoudigde versie van  $B$ 

```

1 if  $B = \text{true}$  of  $B = \text{false}$  then
2   | return  $B$ ;
3 else
4   | if  $B.\text{kern}$  is een kwantificatiekern then
5     | FOBDT in  $B.\text{kern} = \text{simpBDT}(\text{FOBDT in } B.\text{kern})$ ;
6     |  $\varphi = \text{formule die } B.\text{kern} \text{ voorstelt}$ ;
7     | if  $\varphi$  is aritmetisch then
8       |  $\psi = \bigwedge_{\psi_i \in W} \psi_i$ ;
9       | if  $\psi \Rightarrow \varphi$  then
10        | return  $\text{simpBDT}(B.\text{truetak})$ ;
11        | if  $\psi \Rightarrow \neg\varphi$  then
12          | return  $\text{simpBDT}(B.\text{falsetak})$ ;
13      |
14    | if  $\varphi$  is aritmetisch then
15      | voeg  $\varphi$  toe aan  $W$ ;
16      |  $B.\text{truetak} = \text{simpBDT}(B.\text{truetak})$ ;
17    | if  $\varphi$  is aritmetisch then
18      | verwijder  $\varphi$  uit  $W$ ;
19      | voeg  $\neg\varphi$  toe aan  $W$ ;
20      |  $B.\text{falsetak} = \text{simpBDT}(B.\text{falsetak})$ ;
21    | if  $\varphi$  is aritmetisch then
22      | verwijder  $\neg\varphi$  uit  $W$ ;
23    | return  $B$ ;
24
```

---

eerste-ordetheorie  $\mathcal{T} \subset \mathcal{L}_{\mathbf{FO}, \Sigma}$  over een vocabularium

$$\Sigma = \{\Sigma_{type}, \Sigma_{pred}, \Sigma_{func}\} \quad (4.1)$$

$$= \{\{N\}, \quad (4.2)$$

$$\{Gelijk(N, N)\}, \quad (4.3)$$

$$\{S(N) : N, Plus(N, N) : N, Maal(N, N) : N, Nul : N\}\} \quad (4.4)$$

met interpretatie,

$$\mathcal{N} = \langle \mathbb{N}; = (\mathbb{N}, \mathbb{N}); S(\mathbb{N}), +(\mathbb{N}, \mathbb{N}), \cdot(\mathbb{N}, \mathbb{N}), 0 \rangle$$

waarbij  $S$  de successorfunctie is, en  $+$ ,  $\cdot$ ,  $=$  en  $0$  de gewoonlijke interpretatie hebben<sup>3</sup>. Het probleem met Peano is dat het onbeslisbaar is, zoals bewezen werd door Gödel in 1931 ([God31]). Er bestaan natuurlijk beperkingen op deze theorie, die wel beslisbaar zijn, neem bijvoorbeeld de beperking van deze theorie op de taal  $\mathcal{L}_{\mathbf{FO}, \Sigma_{=}}$ , waarbij  $\Sigma_{=}$  enkel bestaande uit de constante  $0$  en de relatie  $=$ . Deze beperking is duidelijk beslisbaar, maar natuurlijk niet erg bruikbaar.

In 1929 bedacht de Poolse wiskundige Mojżesz Presburger echter een andere beperking, die hij Presburger aritmetiek ([Pre29]) doopte. Presburger aritmetiek is de beperking van Peano aritmetiek op  $\mathcal{L}_{\mathbf{FO}, \Sigma_{+}}$ , waarbij  $\Sigma_{+}$  bestaat uit de constante  $0$ , de relatie  $=$ , en de functies  $S(x)$  en  $Plus(x, y)$ . Het is duidelijk dat deze beperking een stuk bruikbaar is dan bijvoorbeeld de beperking op  $\mathcal{L}_{\mathbf{FO}, \Sigma_{=}}$  die we hierboven aangehaald hebben. Presburger bewees echter ook dat zijn aritmetiek beslisbaar was. In dit hoofdstuk wordt deze beslisbaarheid ook bewezen, maar op een andere manier als dit gebeurd is door Presburger zelf. Het bewijs in deze thesis is gedeeltelijk gebaseerd op een bewijs van de bewijsbaarheid van deze theorie in [End72].

**Definitie 4.3.** *Presburger aritmetiek is een theorie  $\mathcal{T}$  in de eerste-ordeloga, gegeven door volgende axioma's: (waarbij we  $Plus(x, y)$  afkorten door  $x + y$  en  $x \neq y$  als afkorting gebruiken voor  $\neg(x = y)$ )*

1.  $\forall x(x + 0 = x)$
2.  $\forall x(S(x) \neq 0)$
3.  $\forall x, y(S(x) = S(y) \rightarrow x = y)$
4.  $\forall x, y(S(x + y) = x + S(y))$
5. *Het inductieaxioma: voor elke formule  $P(x)$ , waar de variabelen  $t_1, t_2, \dots, t_n, x$  vrij voorkomen, hebben we een axioma van de vorm:*

$$\forall t_1, t_2, \dots, t_n[P(0) \wedge \forall x(P(x) \rightarrow P(S(x))) \rightarrow \forall x(P(x))]$$

---

<sup>3</sup>Omdat de notatie licht en leesbaar te houden schrijven we verder natuurlijk gewoon  $a + b$ ,  $a \cdot b$  en  $a = b$  in plaats van  $+(a, b)$ ,  $\cdot(a, b)$ , en  $=(a, b)$ .

We bekijken eerst of deze nieuwe theorie wel enigszins bruikbaar is. Hiervoor moet o.a. gelden dat de theorie consistent is, m.a.w. dat de theorie een model heeft. Dit blijkt zo te zijn, want als we onze standaard interpretatie voor Peano beperken voor onze theorie, dan krijgen we

$$\mathcal{N} = \langle \mathbb{N}; = (\mathbb{N}, \mathbb{N}); S(\mathbb{N}), +(\mathbb{N}, \mathbb{N}), 0 \rangle,$$

en deze blijkt perfect aan deze axioma's te voldoen. Om bruikbaar te zijn, moet deze theorie ook wel genoeg lijken op de theorie voor de natuurlijke getallen, zoals we hem kennen. We vragen ons af of standaard resultaten uit de natuurlijke getallen zoals associativiteit en commutativiteit voor de optelling blijven gelden. We bewijzen als illustratie voor enkele gekende resultaten dat dit inderdaad zo is.

**Propositie 4.2.** *De formule  $\forall x(0 + x = x)$  is een gevolg van de theorie van de Presburger aritmetiek, samen met axioma 1 geeft dit dus de eigenschap dat 0 het neutraal element voor optelling is.*

*Bewijs.* We gebruiken eerst axioma 5, het inductieaxioma, op variabele  $x$ . We moeten nu enkel nog bewijzen dat  $0 + 0 = 0$ , en dat  $0 + x = x \rightarrow 0 + S(x) = S(x)$ . De basisstap volgt direct uit axioma 1. Om de inductiestap te bewijzen gebruiken we eerst dat  $0 + S(x) = S(0 + x)$ , wegens axioma 4, en vervolgens gebruiken we de inductiehypothese, die dan zegt dat  $0 + x = x$ , en dus  $0 + S(x) = S(0 + x) = S(x)$ , wat het lemma bewijst.  $\square$

**Propositie 4.3.** *De formule  $\forall x, y(S(x + y) = S(x) + y)$  is een gevolg van de theorie van de Presburger aritmetiek.*

*Bewijs.* We gebruiken eerst axioma 5, het inductieaxioma, op variabele  $y$ . We moeten nu enkel nog bewijzen dat  $S(x + 0) = S(x) + 0$ , en dat  $S(x + y) = S(x) + y \rightarrow S(x + S(y)) = S(x) + S(y)$ . We bewijzen eerst de basisstap. Er geldt dat  $S(x + 0) = S(x)$ , en dat  $S(x) + 0 = S(x)$ , want ze volgen allebei rechtstreeks uit axioma 1. Deze 2 gelijkheden samen bewijzen de basisstap. Om de inductiestap te bewijzen gebruiken we eerst dat  $S(x + S(y)) = S(S(x + y))$ , wegens axioma 4, vervolgens gebruiken we de inductiehypothese, die dan zegt dat  $S(x + S(y)) = S(S(x + y)) = S(S(x) + y)$ , als we dan nog eens axioma 4 gebruiken, krijgen we,  $S(x + S(y)) = S(S(x) + y) = S(x) + S(y)$ , wat het lemma bewijst.  $\square$

**Propositie 4.4.** *De formule  $\forall x, y(x + y = y + x)$  is een gevolg van de theorie van de Presburger aritmetiek, of met andere woorden de optelling is commutatief.*

*Bewijs.* We gebruiken opnieuw eerst axioma 5, het inductieaxioma, op variabele  $y$ . We moeten nu enkel nog bewijzen dat  $x + 0 = 0 + x$ , en dat  $x + y = y + x \rightarrow x + S(y) = S(y) + x$ . De basisstap volgt uit lemma 4.2 en axioma 1, want deze geven dat  $x + 0 = x = x = 0 + x$ . Voor het bewijs van de inductiestap gebruiken we eerst axioma 4, dat zegt dat  $x + S(y) = S(x + y)$ . Hier laten we dat de inductiehypothese op los, dit geeft ons dan  $x + S(y) = S(x + y) = S(y + x)$ . We sluiten het bewijs voor de inductiestap af met lemma 4.3, welke ons geeft dat  $x + S(y) = S(x + y) = S(y + x) = S(y) + x$ . Dit bewijst de commutativiteit van de optelling.  $\square$



Op dezelfde manier kunnen ook andere bekende resultaten zoals bijvoorbeeld associativiteit aangetoond worden.

### 4.3.2 Beslisbaarheid

Voor we overgaan tot het aantonen van de beslisbaarheid van Presburger aritmetiek, hebben we eerst een belangrijk resultaat over kwantor eliminatie nodig.

**Definitie 4.4.** Een eerste-ordetheorie  $\mathcal{T}$  over een vocabularium  $\Sigma$  laat **kwantor eliminatie** toe als en slechts als voor elke gesloten formule  $F$  er een kwantorvrije gesloten formule  $G$  bestaat zodat  $\mathcal{T} \models F \leftrightarrow G$ .

Volgende propositie geeft ons een criterium waarmee we kunnen bewijzen dat een bepaalde theorie kwantor eliminatie toelaat.

**Propositie 4.5.** Een theorie  $\mathcal{T}$  waarbij er voor elke formule van de vorm  $\exists x (\alpha_0 \wedge \dots \wedge \alpha_n)$  (met elke  $\alpha_i$  een atomische formule, of de negatie ervan) er een kwantorvrije formule  $G$  bestaat zodat  $\mathcal{T} \models \exists x (\alpha_0 \wedge \dots \wedge \alpha_n) \leftrightarrow G$ , laat kwantor eliminatie toe.

*Bewijs.* (Opm: In dit hele bewijs bedoelen we met  $\alpha_i, \beta_i, \dots, \gamma_i$  altijd een atomische formule of de negatie ervan.) Neem een gesloten formule  $F$ , dan kunnen we  $F$  schrijven als

$$Q_1 x_1, Q_2 x_2, \dots, Q_n x_n F'[x_1, \dots, x_n],$$

met  $F'[x_1, \dots, x_n]$  kwantorvrij. We bewijzen dat er een kwantorvrije gesloten formule  $G$  bestaat zodat  $\mathcal{T} \models F \leftrightarrow G$ . Hierbij weten we dat voor elke formule  $H[x_1, \dots, x_n]$  van de vorm  $\exists x (\alpha_0 \wedge \dots \wedge \alpha_n)$  er een kwantorvrije formule  $K[x_1, \dots, x_n]$  bestaat zodat

$$\mathcal{T} \models \{\exists x (\alpha_0 \wedge \dots \wedge \alpha_n) \leftrightarrow K\}.$$

We bewijzen dit per inductie op het aantal kwantoren in  $F$ .

Stel dat  $F$  één kwantor bevat, dan is  $F$  van de vorm  $Q_1 x F'[x]$ , met  $F'[x]$  kwantorvrij. We tonen eerst aan dat het voldoende te bewijzen is dat de basisstap geldt als  $Q_1$  een existentiële kwantor is, want stel dat  $Q_1$  een universele kwantor is, dan is  $F$  van de vorm  $\forall x F'[x]$ , of anders geschreven  $\neg \exists x (\neg F'[x])$ . Als er dan een  $G$  bestaat zodat

$$\mathcal{T} \models (\exists x \neg F'[x]) \leftrightarrow G,$$

dan volgt daaruit rechtstreeks dat

$$\mathcal{T} \models (\forall x F'[x]) \leftrightarrow \neg G.$$

Waarbij natuurlijk het kwantorvrij en gesloten zijn van  $\neg G$  rechtstreeks volgt uit het kwantorvrij en gesloten zijn van  $G$ .

Stel dan nu dat  $Q_1$  een existentiële kwantor is, dan herschrijven we  $\exists x F'[x]$  eerst als

$$\exists x ((\alpha_0 \wedge \dots \wedge \alpha_n) \vee (\beta_0 \wedge \dots \wedge \beta_n) \vee \dots \vee (\gamma_0 \wedge \dots \wedge \gamma_n)).$$

Merk op dat we dit inderdaad altijd mogen doen, want dit is gewoon  $F'$  herschrijven in zijn disjunctieve normaalvorm, aangezien  $F'$  kwantorvrij is. Deze formule is verder nog logisch equivalent met

$$\exists x (\alpha_0 \wedge \dots \wedge \alpha_n) \vee \exists x (\beta_0 \wedge \dots \wedge \beta_n) \vee \dots \vee \exists x (\gamma_0 \wedge \dots \wedge \gamma_n).$$

Er volgt nu uit de hypothese van het lemma dat er kwantorvrije  $\alpha, \beta, \dots, \gamma$  bestaan zodat

$$\begin{aligned} \mathcal{T} &\models \exists x (\alpha_0 \wedge \dots \wedge \alpha_n) \leftrightarrow \alpha \\ &\models \exists x (\beta_0 \wedge \dots \wedge \beta_n) \leftrightarrow \beta \\ &\models \dots \\ &\models \exists x (\gamma_0 \wedge \dots \wedge \gamma_n) \leftrightarrow \gamma \end{aligned}$$

Merk op dat  $\alpha, \beta, \dots, \gamma$  duidelijk ook gesloten zijn, aangezien ze  $x$  niet bevatten, en  $x$  de enige vrije variabele was in  $(\alpha_0 \wedge \dots \wedge \alpha_n), (\beta_0 \wedge \dots \wedge \beta_n), \dots, (\gamma_0 \wedge \dots \wedge \gamma_n)$ . We definiëren nu  $G$  als  $\alpha \vee \beta \vee \dots \vee \gamma$ , dan geldt duidelijk,  $\mathcal{T} \models \{F \leftrightarrow G\}$ . Bijgevolg is de basisstap bewezen.

Stel nu dat  $F$  van de vorm  $Q_1x_1, Q_2x_2, \dots, Q_nx_n F'[x_1, \dots, x_n]$  is. Analoog als in de basisstap is het voldoende de inductiestap te bewijzen als  $Q_n$  een existentiële kwantor is. We herschrijven  $\exists x_n F'[x_n]$  opnieuw (zoals in de basisstap) als

$$\exists x ((\alpha_0 \wedge \dots \wedge \alpha_n) \vee (\beta_0 \wedge \dots \wedge \beta_n) \vee \dots \vee (\gamma_0 \wedge \dots \wedge \gamma_n)).$$

Ook volledig analoog met hierboven vinden we een kwantorvrije formule  $H[x_1, \dots, x_{n-1}]$ , waarvoor  $\mathcal{T} \models \exists x_n F'[x_n] \leftrightarrow H$ . We passen dan de inductiehypothese toe op

$$Q_1x_1, Q_2x_2, \dots, Q_{n-1}x_{n-1} H[x_1, x_2, \dots, x_{n-1}],$$

en we vinden dan een gesloten kwantorvrije formule  $G$  zodat  $\mathcal{T} \models F \leftrightarrow G$ .  $\square$

Om vervolgens de beslisbaarheid van Presburger aritmetiek te bewijzen, gaan we een nieuw vocabularium introduceren, noem  $\Pi$  het vocabularium met type  $N$ , constanten  $Nul, Een$ , unaire predicaten  $Deelbaar_2, Deelbaar_3, Deelbaar_4, Deelbaar_5, \dots$ , het binaire predicaat  $KleinerDan$ , en de binaire functiesymbolen  $Plus$  en  $Min$ . Dus

$$\Pi = \{\Pi_{type}, \Pi_{pred}, \Pi_{func}\} \quad (4.5)$$

$$= \{\{N\}, \quad (4.6)$$

$$\{KleinerDan(N, N), Deelbaar_2(N), Deelbaar_3(N), Deelbaar_4(N), \dots\}, \quad (4.7)$$

$$\{Plus(N, N) : N, Min(N, N) : N, Nul : N, Een : N\} \quad (4.8)$$

en bekijk dan

$$\mathcal{M} = \{\mathbb{Z}; <(\mathbb{Z}, \mathbb{Z}), D_2(\mathbb{Z}), D_3(\mathbb{Z}), D_4(\mathbb{Z}), \dots; +(\mathbb{Z}, \mathbb{Z}), -(\mathbb{Z}, \mathbb{Z}), 0, 1\}$$

als interpretatie voor  $\mathcal{K} = \mathcal{L}_{\mathbf{FO}, \Pi}$ , waarbij de predicaten en de constanten hun gewoonlijke betekenis hebben. Met  $D_2, D_3, D_4, \dots$  bedoelen we deelbaar door respectievelijk 2, 3, 4,  $\dots$ . Om nu te bewijzen dat er een algoritme bestaat om te beslissen of een formule  $F$  uit

$\mathcal{L}_+ = \mathcal{L}_{\mathbf{FO}, \Sigma_+}$  waar is in  $\mathcal{N}$ , vertalen we  $F$  eerst naar een formule  $F'$  in  $\mathcal{K}$ . Dit doen we door elke  $S \setminus 1$  te vervangen door  $+1$ , elke  $r = s$  door  $r < s + 1 \wedge s < r + 1$ , en elke kwantificatie  $\forall x$  of  $\exists x$  te vervangen door een voorwaardelijke kwantificatie:

$$\forall x(((x < 1 \wedge 0 < x + 1) \vee 0 < x) \rightarrow \dots) \quad \exists x(((x < 1 \wedge 0 < x + 1) \vee 0 < x) \wedge \dots).$$

Nu geldt dat  $F$  waar is in  $\mathcal{N}$  als en slechts als  $F'$  waar is in  $\mathcal{M}$ , wegens volgende propositie.

**Propositie 4.6.** *Neem een willekeurige formule  $F$  uit  $\mathcal{L}_+$ , we definiëren  $F'$  door in  $F$  elke  $(S \setminus 1)$  te vervangen door  $+1$ , elke  $r = s$  door  $r < s + 1 \wedge s < r + 1$  en elke kwantificatie te vervangen door een voorwaardelijke kwantificatie, zoals hierboven beschreven. Dan geldt dat  $F'$  een formule is in  $\mathcal{K}$ , en dat  $F$  waar is in  $\mathcal{N}$  als en slechts als  $F'$  waar is in  $\mathcal{M}$ .*

*Bewijs.* Dat  $F'$  een formule is in  $\mathcal{K}$  is duidelijk, we moeten dus enkel de bewering over de waarheid van de formules bewijzen. Voor de successorfunctie en de gelijkheidsrelatie is het te bewijzen triviaal, want de standaardinterpretatie van  $S(x)$  en van  $x + 1$  komen net zoals de standaardinterpretatie van  $r = s$  en  $r < s + 1 \wedge s < r + 1$  duidelijk overeen.

We bewijzen ook nog dat we de universele kwantor mogen vervangen, het bewijs voor de existentiële kwantor verloopt analoog. Neem een formule  $G$  in  $\mathcal{L}_+$  van de vorm  $\forall x G'[x]$ , dan is deze formule waar in  $\mathcal{N}$  als en slechts als  $G'[x]$  waar is voor alle natuurlijke getallen. Neem dan de formule  $\forall x((x < 1 \wedge 0 < x + 1) \vee 0 < x) \rightarrow G'[x]$ , deze formule is waar in  $\mathcal{M}$  als voor alle gehele getallen ofwel geldt dat  $x < 0$  ofwel geldt dat  $G'[x]$  waar is. Of met andere woorden, als  $G'[x]$  waar is voor alle natuurlijke getallen.  $\square$

We bewijzen nu eerst dat de vertaling van Presburger aritmetiek naar  $\mathcal{K}$  volgens propositie 4.6 kwantoreliminatie toestaat.

**Stelling 4.1.** *De vertaling van Presburger aritmetiek naar  $\mathcal{K}$  volgens propositie 4.6 staat eliminatie van kwantoren toe.*

*Bewijs.* We introduceren eerst enkele notationale afspraken. Laat  $t$  een term zijn, en  $n$  een natuurlijk getal, dan schrijven we  $nt$  in plaats van  $t + t + \dots + t$  ( $n$  keer),  $-t$  in plaats van  $0 - t$  en  $-nt$  in plaats van  $0 - t - t - \dots - t$  ( $n$  keer). Verder gebruiken we  $n$  als afkorting voor  $1 + 1 + \dots + 1$  ( $n$  keer). Wegens propositie 4.5 is het voldoende aan te tonen dat kwantoreliminatie mogelijk is voor een formule  $F$  van de vorm  $\exists x (\alpha_0 \wedge \dots \wedge \alpha_n)$  (met elke  $\alpha_i$  een atomische formule, of de negatie ervan). Merk op dat de atomische formules hier enkel  $t < s$  en  $D_i(t)$  zijn, met  $s, t$  termen. Voor we de kwantoren kunnen elimineren, moeten we  $F$  eerst nog wat herschrijven.

Eerst werken we negaties van atomische formules weg. We vervangen  $\neg(t < s)$  door  $s < t + 1$ , en we vervangen  $\neg D_m(t)$  door

$$(D_m(t + 1) \wedge D_m(t + 2) \wedge \dots \wedge D_m(t + m - 1)).$$

Merk op dat de correctheid van deze laatste substitutie volgt uit het feit dat voor elke  $a$  geldt dat  $m$  exact één element deelt van volgende rij  $a, a + 1, a + 2, \dots, a + m - 1$ .

Merk ook op dat  $F$  nog van de vorm  $\exists x (\alpha_0 \wedge \dots \wedge \alpha_n)$  is, maar dat nu elke  $\alpha_i$  een atomische formule is, zonder negatie. We herschrijven verder  $t < s$  door  $0 < (s - t)$ . We gaan nu elke term in een normaalvorm schrijven. We kunnen elke term herschrijven als  $kx, -kx, kx + t, -kx + t$  of  $t$ , met  $t$  een term die  $x$  niet bevat. We substitueren dan ook nog elke formule van de vorm  $0 < -kx$  door  $kx < 0$ ,  $0 < kx + t$  door  $-t < kx$  en  $0 < -kx + t$  door  $kx < t$ . Alle atomische formules zijn nu ofwel van de vorm  $D_m(s)$ , ofwel van de vorm  $t < kx$  of  $kx < t$ , met  $k$  positief, en  $t$  een term die  $x$  niet bevat. We noemen atomische formules van de vorm  $t < kx$  verder kleiner-dan-ongelijkheden, en formules van de vorm  $kx < t$  groter-dan-ongelijkheden. Uit lemma 4.1.1 hieronder volgt nu dat we  $F$  kunnen herschrijven zodat elk disjunct maar 1 kleiner-dan- en 1 groter-dan-ongelijkheid bevat. We gebruiken hierna lemma 4.1.2 hieronder. Als resultaat krijgen we dan dat er in elk disjunct maximaal één deelbaarheidspredicaat, één kleiner-dan-ongelijkheid, en één groter-dan-ongelijkheid voorkomt.

Omdat we de existentiële kwantor kunnen propageren in de disjunctie naar elk van de disjuncten, en deze dan kunnen beperken tot het deel van die conjunctie dat  $x$  bevat, blijft de eliminatie van kwantoren beperkt tot volgende 7 gevallen:

Geval	Herschrijven als	Argument
$\exists x(s < jx)$	<b>t</b>	voor alle constanten $s, j$ bestaat er een getal in $\mathbb{Z}$ dat groter is dan $s$ en deelbaar is door $j$
$\exists x(kx < t)$	<b>t</b>	voor alle constanten $t, k$ bestaat er een getal in $\mathbb{Z}$ dat kleiner is dan $t$ en deelbaar is door $k$
$\exists D_m(x - i)$	<b>t</b>	voor elke gegeven deler $m$ en rest $i$ bestaat er een getal $a$ zodat $a \equiv i \pmod{m}$
$\exists x(D_m(x - i) \wedge s < jx)$	<b>t</b>	voor elke gegeven deler $m$ en rest $i$ bestaat er een willekeurig groot getal $a$ zodat $a \equiv i \pmod{m}$
$\exists x(D_m(x - i) \wedge kx < t)$	<b>t</b>	voor elke gegeven deler $m$ en rest $i$ bestaat er een willekeurig klein getal $a$ zodat $a \equiv i \pmod{m}$
$\exists x(kx < t \wedge s < jx)$	$\exists x(D_{jk}(x - 0) \wedge ks < x \wedge x < jt)$	Zie hieronder
$\exists x(D_m(x - i) \wedge s < jx \wedge kx < t)$	$\exists x(D_{jkm}(x - jki) \wedge ks < x \wedge x < jt)$	Zie hieronder

We moeten dus geval 6 en 7 nog uitwerken. Merk voor geval 6 op dat  $\exists(kx < t \wedge s < jx)$  equivalent is met  $\exists x(jkx < jt \wedge ks < jkx)$ , en dat als we  $jkx$  nu hernoemen als  $x$ , we volgende formule krijgen:  $\exists x(x < jt \wedge ks < x \wedge D_{jk}(x - 0))$ . Geval 7 is ongeveer analoog, we krijgen op dezelfde manier dat  $\exists x(D_m(x - i) \wedge s < jx \wedge kx < t)$  equivalent is met  $\exists x(D_{jkm}(jkx - jki) \wedge ks < jkx \wedge jkx < jt)$ , welke op zijn beurt analoog equivalent is met  $\exists x(D_{jkm}(y - jki) \wedge ks < y \wedge y < jt)$ .

Er blijft nu nog een laatste kleine stap over,  $x$  is namelijk nog altijd niet uit onze formule,

maar de zinnen die  $x$  bevatten, hebben wel allemaal dezelfde vorm, namelijk:

$$\exists x(D_m(x - i) \wedge s < x \wedge x < t). \quad (4.9)$$

Deze kunnen we vervangen door

$$(D_m(s+1-i) \wedge s+1 < t) \vee (D_m(s+2-i) \wedge s+2 < t) \vee \dots \vee (D_m(s+m-i) \wedge s+m < t). \quad (4.10)$$

Dit is correct, want stel dat (4.9) geldt, dan bestaat er een  $x$  groter dan  $s$ , zodat  $x \equiv i \pmod{m}$ , dus moet  $x$  bevat zijn tussen 1 en  $m$ , en dit is exact wat (4.10) zegt. De omgekeerde implicatie is triviaal.

Dit bewijst dus dat de theorie van  $\mathcal{K}$  eliminatie van kwantoren toestaat.  $\square$

**Lemma 4.1.1.** *Voor elke gesloten formule  $F$ , kunnen we  $F$  herschrijven in disjunctieve normaalvorm, waarbij elk disjunct maximaal 1 kleiner-dan-ongelijkheid en 1 groter-dan-ongelijkheid bevat.*

*Bewijs.* We bewijzen het lemma enkel voor kleiner-dan-ongelijkheden, het bewijs voor groter-dan-ongelijkheden is analoog (we moeten wel aantonen dat het herschrijven voor groter-dan-ongelijkheden geen nieuwe kleiner-dan-ongelijkheden introduceert). Neem een gesloten formule  $F$  en schrijf deze in zijn disjunctieve normaalvorm. Dan geldt voor elk disjunct van  $F$  dat dit disjunct ofwel één of geen kleiner-dan-ongelijkheden bevat, of dat er een conjunctie in staat van de vorm  $t_1 < k_1x \wedge t_2 < k_2x$ . In het eerste geval, valt er natuurlijk niets te bewijzen, dus stel dat we ons in het tweede geval bevinden. Het is nu voldoende aan te tonen dat we dit disjunct kunnen herleiden tot een nieuwe formule in disjunctieve normaalvorm, waarbij elk disjunct strikt minder kleiner-dan-ongelijkheden bevat, want dan kunnen we door deze stap eventueel een eindig aantal keer te herhalen elke formule herschrijven tot een formule in disjunctieve normaalvorm waarbij elk disjunct maximaal één kleiner-dan-ongelijkheid bevat.

Stel dus dat we een disjunct hebben van de vorm

$$a_0 \wedge \dots \wedge t_1 < k_1x \wedge t_2 < k_2x \wedge \dots \wedge a_n.$$

Als we kunnen aantonen dat dat we  $t_1 < k_1x \wedge t_2 < k_2x$  mogen vervangen door de disjunctie van (we gebruiken  $k_1t_2 = k_2t_1$  even terug als afkorting voor  $k_1t_2 < k_2t_1 + 1 \wedge k_2t_1 < k_1t_2 + 1$ , voor de leesbaarheid)

$$t_1 < k_1x \wedge k_1t_2 < k_2t_1 \quad (4.11a)$$

$$t_1 < k_1x \wedge k_1t_2 = k_2t_1 \quad (4.11b)$$

$$t_2 < k_2x \wedge k_2t_1 < k_1t_2, \quad (4.11c)$$

dan gebruiken we de distributiviteitseigenschap van de eerste-orde logica, en mogen we ons disjunct vervangen door een disjunctie van

$$\begin{aligned} a_0 \wedge \dots \wedge t_1 < k_1x \wedge k_1t_2 < k_2t_1 \wedge \dots \wedge a_n \\ a_0 \wedge \dots \wedge t_1 < k_1x \wedge k_1t_2 = k_2t_1 \wedge \dots \wedge a_n \\ a_0 \wedge \dots \wedge t_2 < k_2x \wedge k_2t_1 < k_1t_2 \wedge \dots \wedge a_n. \end{aligned} \quad (4.12)$$

Het is duidelijk dat deze nieuwe disjuncten allemaal strikt minder kleiner-dan-ongelijkheden bevatten. Er rest ons enkel nog de substitutie van  $t_1 < k_1x \wedge t_2 < k_2x$  te verantwoorden. Stel dat  $t_1 < k_1x \wedge t_2 < k_2x$  geldt, dan is duidelijk dat 4.11a, 4.11b of 4.11c ook geldt, aangezien voor elke van deze 3 het eerste deel volgt uit  $t_1 < k_1x \wedge t_2 < k_2x$  en de 3 laatste delen complementair zijn. Stel dat omgekeerd bijvoorbeeld 4.11a geldt, dat moeten we nog bewijzen dat  $t_2 < k_2x$ . Dit volgt meteen als we de vergelijkingen een beetje manipuleren:

$$\begin{aligned} t_1 &< k_1x && \text{vergelijking 4.11a} \\ k_2t_1 &< k_2k_1x && \text{beide zijden } \cdot k_2 \\ k_1t_2 &< k_2k_1x && \text{LHS } k_1t_2 < k_2t_1 \\ t_2 &< k_2x \end{aligned}$$

Het bewijs voor vergelijking 4.11b en 4.11c verloopt analoog. Het is duidelijk dat het herschrijven van  $t_1 < k_1x \wedge t_2 < k_2x$ , geen nieuwe termen die  $x$  bevatten creëert. Dit geldt analoog voor het herschrijven van groter-dan-ongelijkheden.  $\square$

**Lemma 4.1.2.** *Neem een gesloten formule  $F$  in disjunctieve normaalvorm, waarin elk disjunct maximaal één kleiner-dan-ongelijkheid en één groter-dan-ongelijkheid bevat, dan kunnen we  $F$  herschrijven in disjunctieve normaalvorm, waarbij elk disjunct maximaal één kleiner-dan-ongelijkheid, één groter-dan-ongelijkheid, en één deelbaarheidspredicaat bevat.*

*Bewijs.* Met behulp van lemma 4.1.3, kunnen we aannemen dat elk deelbaarheidspredicaat in  $F$  van de vorm  $D_{p^e}(x - i)$  is, met  $i$  een natuurlijk getal,  $0 \leq i < p^e$ , en  $p^e$  de macht van een priemgetal. We schrijven  $F$  om te beginnen in zijn disjunctieve normaalvorm, en we nemen een willekeurig disjunct  $F'$  van  $F$ , dan moeten we bewijzen dat we  $F'$  kunnen herschrijven tot een formule in disjunctieve normaalvorm, waarbij elk disjunct maximaal 1 deelbaarheidspredicaat, één kleiner-dan-ongelijkheid en één groter-dan-ongelijkheid bevat. We veronderstellen dus dat  $F'$  meerdere deelbaarheidspredicaten bevat, anders is het te bewijzen triviaal. Dus schrijf

$$F' = a \wedge D_{p_1^{e_1}}(x - i_1) \wedge D_{p_2^{e_2}}(x - i_2) \wedge \cdots \wedge D_{p_k^{e_k}}(x - i_k),$$

met  $a$  een conjunctie die geen deelbaarheidspredicaat, en maximaal één kleiner-dan-ongelijkheid en één groter-dan-ongelijkheid bevat. Met  $p_1, \dots, p_k$  bedoelen we opnieuw priemgetallen.

Stel nu dat  $F'$  twee deelbaarheidspredicaten bevat,  $D_{p_a^{e_a}}(x - i_a)$ ,  $D_{p_b^{e_b}}(x - i_b)$ , en dat  $p_a = p_b$ . Dan vervangen we (stel zonder verlies van algemeenheid dat  $e_a \leq e_b$ ):

$$D_{p_a^{e_a}}(x - i_a) \wedge D_{p_b^{e_b}}(x - i_b) \tag{4.13}$$

door

$$D_{p_a^{e_a}}(i_b - i_a) \wedge D_{p_b^{e_b}}(x - i_b) \tag{4.14}$$

We moeten nog bewijzen dat deze substitutie gerechtvaardigd is. We merken eerst op dat omdat  $e_a \leq e_b$  er geldt dat  $D_{p_a^{e_a}}(x - i_b)$  volgt uit  $D_{p_b^{e_b}}(x - i_b)$ . Vergelijking (4.13) is

duidelijk equivalent met:

$$\begin{aligned} x &\equiv i_a \pmod{p_a^{e_a}} \\ x &\equiv i_b \pmod{p_a^{e_a}} \\ x &\equiv i_b \pmod{p_b^{e_b}} \end{aligned} \tag{4.15}$$

Vergelijking (4.14) is op zijn beurt duidelijk equivalent met:

$$\begin{aligned} i_b &\equiv i_a \pmod{p_a^{e_a}} \\ i_b &\equiv x \pmod{p_a^{e_a}} \\ i_b &\equiv x \pmod{p_b^{e_b}} \end{aligned} \tag{4.16}$$

Het equivalent zijn van (4.13) en (4.14) volgt nu rechtstreeks uit het equivalent zijn van (4.15) en (4.16).

Door deze redenering een eindig aantal keer toe te passen, weten we dat we  $F'$  kunnen herschrijven tot een disjunct van volgende vorm:

$$a \wedge b \wedge D_{p_1^{e_1}}(x - i_1) \wedge D_{p_2^{e_2}}(x - i_2) \wedge \cdots \wedge D_{p_k^{e_k}}(x - i_k),$$

met alle  $p_j$  verschillend, en  $b$  een conjunctie van deelbaarheidspredicaten die geen  $x$  bevatten. Wat hier nu staat is equivalent met volgend stelsel vergelijkingen:

$$\begin{aligned} x &\equiv i_1 \pmod{p_1^{e_1}} \\ x &\equiv i_2 \pmod{p_2^{e_2}} \\ x &\equiv i_3 \pmod{p_3^{e_3}} \\ &\equiv \cdots \\ x &\equiv i_k \pmod{p_k^{e_k}} \end{aligned}$$

Volgens de Chinese reststelling bestaat er nu een unieke  $i$  zodat

$$x \equiv i \pmod{m} \quad m = p_1^{e_1} \cdot p_2^{e_2} \cdots p_n^{e_n}.$$

Dus kunnen we schrijven dat

$$F' = a \wedge D_m(x - i) \wedge b.$$

Merk op dat  $a$  niet veranderd is, en dat dus  $F'$  nog steeds maximaal één kleiner-dan en één groter-dan-ongelijkheid bevat. Dit bewijst het lemma.  $\square$

**Lemma 4.1.3.** *Neem een gesloten formule  $F$  in disjunctieve normaalvorm, waarin elk disjunct maximaal één kleiner-dan-ongelijkheid en één groter-dan-ongelijkheid bevat, dan kunnen we  $F$  herschrijven in disjunctieve normaalvorm, waarbij elk disjunct nog steeds maximaal één kleiner-dan-ongelijkheid en één groter-dan-ongelijkheid bevat, en zodat elk deelbaarheidspredicaat in  $F$  dat  $x$  bevat, van de vorm  $D_{p^e}(x - i)$  is, met  $i$  een natuurlijk getal,  $0 \leq i < p^e$ , en  $p^e$  de macht van een priemgetal.*

*Bewijs.* Neem een disjunct  $F'$  van  $F$ , dan geldt:

$$F' = a \wedge D_m(t)$$

met  $a$  een conjunctie die maximaal één kleiner-dan-ongelijkheid en één groter-dan-ongelijkheid bevat, en  $t$  een term die  $x$  bevat. We zullen dit lemma dan bewijzen door effectief  $D_m(t)$  te herschrijven tot een gesloten formule waarin elk deelbaarheidspredicaat van de vorm  $D_{p^e}(x - i)$  is, die geen ongelijkheden bevat. Dit bewijst dan het lemma aangezien we deze redenering voor elk deelbaarheidspredicaat in  $F'$  kunnen toepassen, en het in disjunctieve normaalvorm schrijven geen termen creëert (en er natuurlijk niet twee keer dezelfde term in een disjunct staat).

**Stap 1: Elk deelbaarheidspredicaat is van de vorm  $D_m(kx - t)$ :** In stelling 4.1 hebben we bewezen dat elke term die een variabele  $x$  bevat te schrijven is als  $kx, -kx, kx+t$  of  $-kx+t$ , met  $k$  een natuurlijk getal, en  $t$  een term die  $x$  niet bevat. Volgende tabel zegt ons nu dus dat elk deelbaarheidspredicaat van de vorm  $D_m(kx - t)$  is:

Geval	Herschrijven als	Argument
$D_m(kx)$	$D_m(kx - 0)$	
$D_m(-kx)$	$D_m(kx - 0)$	$m$ deelt $a$ als en slechts als $m$ deelt $-a$
$D_m(kx + t)$	$D_m(kx - (-t))$	
$D_m(-kx + t)$	$D_m(kx - t)$	$m$ deelt $a$ als en slechts als $m$ deelt $-a$

**Stap 2: Elk deelbaarheidspredicaat is van de vorm  $D_m(kx - i)$ :** Stel nu dat  $D_m(kx - t)$  waar is, dan geldt dus dat  $kx - t$  deelbaar is door  $m$ , en dus dat de rest van  $kx$  en de rest van  $t$  bij deling door  $m$  dezelfde is. Met andere woorden, er bestaat een  $i$ ,  $0 \leq i < m$ , zodat  $m|(kx-i)$ , en  $m|(t-i)$ . Dit kunnen we schrijven als  $\exists i(D_m(kx-i) \wedge D_m(t-i))$ , maar omdat de grenzen voor  $i$  eindig zijn, kunnen we dit ook schrijven als de disjunctie van

$$\begin{aligned}
& D_m(kx - 0) \wedge D_m(t - 0) \\
& D_m(kx - 1) \wedge D_m(t - 1) \\
& D_m(kx - 2) \wedge D_m(t - 2) \\
& \dots \\
& D_m(kx - (m-1)) \wedge D_m(t - (m-1))
\end{aligned}$$

Dus is elk deelbaarheidspredicaat dat  $x$  bevat in onze formule van de vorm  $D_m(kx - i)$ , met  $0 \leq i < m$ .

**Stap 3: Elk deelbaarheidspredicaat is van de vorm  $D_m(x - j)$ :** Noem  $j_1, \dots, j_n$  alle waarden voor  $j$ ,  $0 \leq j < m$ , die voldoen aan  $D_m(kj - i)$ . Dan beweren we dat  $D_m(kx - i)$  equivalent is met  $D_m(x - j_1) \vee D_m(x - j_2) \vee \dots \vee D_m(x - j_n)$ . Stel om dit te bewijzen dat  $D_m(kx - i)$  geldt, en noem  $l$  de rest van  $x$  bij deling door  $m$ , dan geldt natuurlijk  $D_m(x - l)$ , dus moeten we nog aantonen dat  $D_m(kl - i)$  waar is. Er geldt dat:

$$\begin{aligned}
kl - i &\equiv kx - i \pmod{m} && \text{want } l \equiv x \pmod{m} \\
&\equiv 0 \pmod{m} && \text{want } D_m(kx - i)
\end{aligned}$$



Stel dan omgekeerd dat  $D_m(x - j_1) \vee D_m(x - j_2) \vee \cdots \vee D_m(x - j_n)$  geldt, voor alle  $j_a$  waarvoor  $D_m(kj_a - i)$  waar is, dan bestaat er natuurlijk een  $b$  zodat  $D_m(x - j_b) \wedge D_l(kj_b - i)$  geldt. Dan volgt:

$$\begin{aligned} kx - i &\equiv kj_b - i \pmod{m} && \text{want } x \equiv j_b \pmod{m} \\ &\equiv 0 \pmod{m} && \text{want } D_m(kj_b - i) \end{aligned}$$

Hiermee hebben we bewezen dat er  $j_1, \dots, j_n$  bestaan zodat elk deelbaarheidspredicaat van de vorm  $D_m(kx - i)$  equivalent is met  $D_m(x - j_1) \vee D_m(x - j_2) \vee \cdots \vee D_m(x - j_n)$ . We kunnen dus  $F$  herschrijven zodat elk deelbaarheidspredicaat van de vorm  $D_m(x - j)$  is, met  $j$  een natuurlijk getal,  $0 \leq j < m$ .

**Stap 4: Elk deelbaarheidspredicaat is van de vorm  $D_{p^e}(x - j)$  (met  $p$  priem):** We schrijven  $m = p_1^{e_1} \cdot p_2^{e_2} \cdots p_n^{e_n}$ , de priemontbinding van  $m$ . Dan geldt dat  $m \mid (x - j)$ , als en slechts als  $(p_1^{e_1} \cdot p_2^{e_2} \cdots p_n^{e_n}) \mid (x - j)$ , als en slechts als  $p_1^{e_1} \mid (x - j) \wedge p_2^{e_2} \mid (x - j) \wedge \cdots \wedge p_n^{e_n} \mid (x - j)$ , omdat  $p_1^{e_1}, p_2^{e_2}, \dots, p_n^{e_n}$  geen gemeenschappelijke factoren bezitten. Dus geldt dat  $D_m(x - j)$  equivalent is met

$$D_{m_1}(x - j) \wedge D_{m_2}(x - j) \wedge \cdots \wedge D_{m_k}(x - j), \text{ waarbij } m = m_1 m_2 m_3 \dots m_k = p_1^{e_1} p_2^{e_2} \dots p_k^{e_k} \text{ de priemontbinding is van } m.$$

Dit bewijst stap 4. Samen met de vorige stappen hebben we dus bewezen dat we  $F$  kunnen herschrijven zodat elk deelbaarheidspredicaat van de vorm  $D_m(x - j)$  is, met  $0 \leq j < m$ , en  $m$  de macht van een priemgetal.

Aangezien elke stap een deelbaarheidspredicaat herschrijft als een disjunctie van conjuncties van deelbaarheidspredicaten, volgt het onmiddellijk dat er geen ongelijkheden gecreëerd worden. Dit bewijst het lemma.  $\square$

Uit stelling 4.1 volgt nu rechtstreeks de beslisbaarheid van Presburger, want neem een formule  $F$  in  $\mathcal{N}$ , dan is  $F$  waar in  $\mathcal{N}$  als en slechts als  $F'$  (zoals boven stelling 4.1 geconstrueerd) waar is in  $\mathcal{M}$ . Dus het beslissingsprobleem van  $F$  herleidt zich tot het beslissingsprobleem van  $F'$  in  $\mathcal{M}$ . Deze  $F'$  in  $\mathcal{M}$  kunnen we wegens vorige stelling herschrijven als een kwantorvrije formule  $F''$ . Een atomische formule is beslisbaar, en dus is een kwantorvrije formule als booleaanse combinatie van atomische formules ook beslisbaar. Dus is het beslissen van de waarheid van  $F'$  in  $\mathcal{M}$  en bijgevolg het beslissen van de waarheid van  $F$  in  $\mathcal{N}$  effectief mogelijk.

We kunnen dit resultaat ook veralgemenen voor de theorie  $\mathcal{H}$  van de gehele getallen ( $\mathbb{Z}$ ), met relatie  $</2$ , functies  $+/2$ ,  $-/2$  en constanten  $0, 1$ , die geen vermenigvuldiging bevat. Voor deze theorie wordt in de literatuur ook vaak de term Presburger aritmetiek gebruikt. Neem namelijk een formule  $G$  in  $\mathcal{H}$ , dan geldt dat  $G$  ook in formule is in  $\mathcal{K}$ . Een rechtstreekse aanpassing van propositie 4.6 geeft ons dan dat  $G$  waar is in de standaardinterpretatie van  $\mathcal{H}$  als en slechts als  $G$  waar is in de standaardinterpretatie van  $\mathcal{K}$  ( $\mathcal{M}$ ). Analoog aan hierboven kunnen we dus  $G$  herschrijven als een kwantorvrije

formule waarvoor dus het beslissen van de waarheid mogelijk is. We gebruiken verder dus deze “versie” van Presburger aritmetiek.

### 4.3.3 Beslissen van Presburger redundantie in de praktijk

Aangezien het dus onmogelijk is om aritmetische redundantie in een FOBDT volledig op te sporen, definiëren we een nieuw soort redundantie, die we wel kunnen opsporen, Presburger redundantie:

**Definitie 4.5.** *We noemen een knoop  $K$  met formule  $\varphi$  **Presburger redundant** als er een pad  $P$  bestaat met padformuleverzameling  $V$  bestaat waarvoor geldt dat:*

$$\bigwedge_{\psi_i \in W} \psi_i \Rightarrow \varphi \quad \text{of} \quad \bigwedge_{\psi_i \in W} \psi_i \Rightarrow \neg\varphi$$

waar is in elk model dat voldoet aan de Presburger axioma's<sup>4</sup>. Hierbij bedoelen we met  $W \subset V$  de verzameling Presburger formules in  $V$ .

Met enkel het theoretische resultaat dat Presburger aritmetiek beslisbaar is, zijn we natuurlijk niet veel, we willen het in de praktijk kunnen gebruiken. In de context van deze thesis werd de Omega-library uit Omega+ ([Che09]) gebruikt. Omega+ is een verbetering van het Omega-project, een pakket dat origineel ontwikkeld werd in de context van software-verificatie. Het werd ontwikkeld begin jaren '90, aan de universiteit van Maryland (US). Het Omega-project bevat buiten de Omega-library ook nog de Omega-calculator, wat een text-based interface is om de Omega-library te kunnen gebruiken, en de Uniform-library en Petit, welke echt specifiek toepassingen zijn voor software verificatie. De Omega-library is een verzameling C++-klassen, die bedoeld zijn om Presburger verzamelingen te kunnen gebruiken en manipuleren. Een Presburger verzameling voor een Presburger formule  $\varphi[x_1, \dots, x_n]$  definiëren we als de deelverzameling van  $\mathbb{Z}^n$ , van alle  $n$ -tupels die voldoen aan  $\varphi[x_1, \dots, x_n]$ . De functionaliteit van het systeem is dus breder dan wat wij ervan benut hebben in de context van deze thesis, wij gebruiken het systeem namelijk enkel als beslisser voor Presburger aritmetiek. Het Omega-project zelf is zoals vermeld al meer dan 15 jaar oud, en is al jaren niet meer onderhouden. Het systeem bevatte nog een heel aantal fouten, en had problemen met verschillende speciale gevallen. In 2009 werden aan de universiteit van Utah (US) veel van deze onvolmaaktheden uit het systeem gehaald, en de verbeterde versie van het systeem werd Omega+ gedoopt.

Het testen of een Presburger-formule  $\varphi$  met vrije variabelen  $x_1, x_2, \dots, x_n$  al dan niet satisfieerbaar is, doen we in dus Omega+ door een verzameling aan te maken van de vorm  $\{x_1, x_2, \dots, x_n | \varphi\}$  en te testen of deze verzameling leeg is. De manier waarop Omega+ dit beslist, gebeurt gelijkaardig aan de manier uit het bewijs in hoofdstuk 4.3.1, namelijk door middel van kwantoreliminatie ([BW03]). De concrete methode is dan wel gelijkaardig aan ons bewijs, maar natuurlijk niet helemaal gelijk. Het bewijs in deze thesis is geschreven met het oog op duidelijkheid en overzichtelijkheid, terwijl de manier waarop Omega+ het doet gebaseerd is op de doelstelling van efficiëntie.

<sup>4</sup>Zie hoofdstuk 4.3.1 voor deze axioma's.

Als we dan terugkijken naar algoritme 4.1, dan zouden we de test of  $K$  (met Presburger formule  $\varphi$ ) redundant is, kunnen uitvoeren door een Omega+ verzameling  $S_0$  voor  $\varphi \wedge \bigwedge_{\psi_i \in V} \psi_i$  en een verzameling  $S_1$  voor  $\neg\varphi \wedge \bigwedge_{\psi_i \in V} \psi_i$  aan te maken. Als dan  $S_0$  leeg is, geldt dus dat  $\varphi \wedge \bigwedge_{\psi_i \in V} \psi_i$  niet satisfieerbaar is, oftewel dat  $\bigwedge_{\psi_i \in V} \psi_i \Rightarrow \neg\varphi$ . Als analoog  $S_1$  leeg is kunnen we besluiten dat  $\bigwedge_{\psi_i \in V} \psi_i \Rightarrow \varphi$ . We vervangen dus de padformuleverzameling van hoofdstuk 4.2 door een Omega+ verzameling  $S$  waaraan we dan voorwaarden en variabelen kunnen toevoegen, en waarmee we de testen kunnen uitvoeren. Hoe we deze formules toevoegen wordt uitgewerkt in algoritme 4.1 in het hoofdstuk over implementatie (hoofdstuk 4.7).

## 4.4 Uiteindelijk algoritme

In dit hoofdstuk worden alle stukjes die in de vorige hoofdstukken besproken zijn, samen genomen om een werkend algoritme te construeren. We maken een algoritme om een FOBDT  $B$  te vereenvoudigen, met de ideeën uit hoofdstuk 4.2, waarbij we de beslissing of een knoop  $K$  redundant is, uitvoeren zoals we uitgewerkt hebben in hoofdstuk 4.3<sup>5</sup>. Dit wil dus zeggen dat we onze padformuleverzameling  $V$  vervangen door een Omega+ verzameling  $S$ . Op een bepaalde knoop  $K$  bevat  $S$  alle  $n$ -tupels die voldoen aan de formule die het pad naar  $K$  voorstelt (en  $n$  vrije variabelen heeft). Als  $K$  een Presburger formule  $\varphi$  bevat, kunnen we testen of  $K$  Presburger redundant is door twee nieuwe Omega+ verzamelingen

$$S_0 = \text{BreidUitEnVoegVoorwToe}(S, \varphi) \quad S_1 = \text{BreidUitEnVoegVoorwToe}(S, \neg\varphi)$$

aan te maken<sup>6</sup>. In het geval dat  $S_0$  of  $S_1$  leeg is, kunnen we  $B$  vervangen door zijn false-respectievelijk truetak.

Als  $S_0$  en  $S_1$  beiden niet leeg zijn, is de kern van  $B$  dus geen redundante knoop. We maken dan een back-up van  $S$ , voegen de formule in de kern (noem die  $\varphi$ ) toe aan  $S$  als het een Presburger formule is met algoritme 4.5, en vereenvoudigen de truetak van  $B$ . Als dit gedaan is, herstellen we de back-up, voegen we  $\neg\varphi$  toe, en vereenvoudigen de falsetak. Hierna geven we dan de vereenvoudigde FOBDT terug. Het algoritme *simpBDT1* staat uitgewerkt in pseudocode in algoritme 4.2.

## 4.5 Voorbeeld

Om de werking van het algoritme te illustreren, werken we hier een voorbeeld uit. We nemen een concrete FOBDT (figuur 4.2a), en we overlopen de werking van het algoritme op deze FOBDT. Omdat het een recursief algoritme is, bekijken we de werking van het algoritme op elke (deel-)FOBDT, en houden we de diepte bij.

<sup>5</sup>Dit impliceert ook dat we nu niet meer alle aritmetische formules bekijken, maar enkel naar Presburger formules

<sup>6</sup>Zie hoofdstuk 4.7.1 voor het algoritme dat formules toevoegt aan een Omega+ verzameling.

---

**Algoritme 4.2:** simpBDT1: aritmetische vereenvoudiging van FOBDTs (tweede versie)

---

**Input:** Een FOBDT  $B$   
**Interne variabelen:** Omega+ verzameling  $S$   
**Output:** Aritmetisch vereenvoudigde versie van  $B$

```

1 if  $B = \text{true}$  of  $B = \text{false}$  then
2   | return  $B$ ;
3 else
4   | if  $B.\text{kern}$  is een kwantificatiekern then
5     | FOBDT in  $B.\text{kern} = \text{simpBDT1}(\text{FOBDT in } B.\text{kern})$ ;
6     |  $\varphi = \text{formule die } B.\text{kern} \text{ voorstelt}$ ;
7     | if  $\varphi$  is een Presburger formule then
8       |  $S' = \text{BreidUitEnVoegVoorwToe}(S, \neg\varphi)$ ;
9       | if  $S' = \text{leeg}$  then
10        | return  $\text{simpBDT1}(B.\text{truetak})$ ;
11        |  $S'' = \text{BreidUitEnVoegVoorwToe}(S, \varphi)$ ;
12        | if  $S'' = \text{leeg}$  then
13          | return  $\text{simpBDT1}(B.\text{falsetak})$ ;
14      |
15      |  $S_{\text{backup}} = S$ ;                                // Maak een back-up van  $S$ 
16      | if  $\varphi$  is een Presburger formule then
17        |  $S = S.\text{BreidUitEnVoegVoorwToe}(\varphi)$ ;
18        |  $B.\text{truetak} = \text{simpBDT1}(B.\text{truetak})$ ;
19        |  $S = S_{\text{backup}}$ ;                                // Herstel back-up
20        | if  $\varphi$  is een Presburger formule then
21          |  $S = S.\text{BreidUitEnVoegVoorwToe}(\neg\varphi)$ ;
22          |  $B.\text{falsetak} = \text{simpBDT1}(B.\text{falsetak})$ ;
23          |  $S = S_{\text{backup}}$ ;                                // Herstel back-up
24      | return  $B$ ;
25
```

---

1. Het algoritme bezoekt eerst de FOBDT in figuur 4.2a, op diepte 0. Hiervoor geldt dat

- $S = \emptyset$
- De FOBDT is niet **t** of **f**
- De kern is geen kwantificatiekern
- De kern bevat een Presburger formule  $x > 2$

We maken dus een verzameling  $S' = \{x | \neg(x > 2)\}$ , en een verzameling  $S'' = \{x | x > 2\}$  en zien dat beide verzamelingen niet leeg zijn. We voegen dus  $x > 2$  toe aan  $S$ , en bezoeken de truetak.

2. De truetak van deze FOBDT bevindt zich we op diepte 1, en is te zien in figuur 4.2b. Er geldt voor deze FOBDT dat

- $S = \{x | x > 2\}$
- De FOBDT is niet **t** of **f**
- De kern is geen kwantificatiekern
- De kern bevat een Presburger formule  $y + 2 > 0$

We maken dus een verzameling  $S' = \{x, y | x > 2 \wedge \neg(y + 2 > 0)\}$ , en een verzameling  $S'' = \{x, y | x > 2 \wedge y + 2 > 0\}$  en zien dat beide verzamelingen niet leeg zijn. We voegen dus  $y + 2 > 0$  toe aan  $S$ , en bezoeken de truetak.

3. Deze FOBDT zit op diepte 2, en vinden we in figuur 4.2c. We weten dat

- $S = \{x | x > 2 \wedge y + 2 > 0\}$
- De FOBDT is niet **t** of **f**
- De kern is geen kwantificatiekern
- De kern bevat een Presburger formule  $x + y = 0$

We maken dus een verzameling  $S' = \{x, y | x > 2 \wedge y + 2 > 0 \wedge \neg(x + y = 0)\}$ , en een verzameling  $S'' = \{x, y | x > 2 \wedge y + 2 > 0 \wedge x + y = 0\}$ . We merken nu op dat de verzameling  $S'' = \{x, y | x > 2 \wedge y + 2 > 0 \wedge x + y = 0\}$  leeg is, er bestaat namelijk geen enkel koppel gehele getallen  $x, y$  dat voldoet aan de voorwaarde in  $S''$ . We vervangen dus deze FOBDT door de FOBDT **f**, en keren terug naar diepte 1.

4. De FOBDT uit stap 2, is nu herwerkt, want één van zijn deelbomen is gewijzigd. We zien de nieuwe FOBDT in figuur 4.2d. We voegen dan de  $\neg y + 2 > 0$  toe aan  $S$ , en bezoeken de falsetak. Maar deze tak is de triviale FOBDT **t**. We keren dus terug naar diepte 0.
5. De FOBDT uit stap 1 is nu ook gewijzigd, en is de FOBDT in figuur 4.2e geworden. We voegen  $\neg x > 2$  toe aan  $S$ , en bezoeken de falsetak.
6. Deze falsetak zit dus op diepte 1, en ziet eruit als in figuur 4.2f. We weten dat

- $S = \{x | \neg x > 2\}$

- De FOBDT is niet **t** of **f**
- De kern is een kwantificatiekern

We bevinden ons dus in het speciale geval waarin we eerst de FOBDT in de kwantificatiekern moeten vereenvoudigen.

7. De kwantificatiekern zien we ook in figuur 4.2f. Over de FOBDT in deze kern weten we dat

- $S = \{x \mid \neg x > 2\}$
- De FOBDT is niet **t** of **f**
- De kern is geen kwantificatiekern
- De kern bevat een Presburger formule  $a > 2$

We maken opnieuw een verzameling  $S' = \{x, a \mid \neg x > 2 \wedge \neg a > 2\}$ , en een verzameling  $S'' = \{x, a \mid \neg x > 2 \wedge a > 2\}$  en zien dat beide verzamelingen niet leeg zijn. We voegen dus  $a > 2$  toe aan  $S$  en bezoeken de truetak.

8. De truetak zit op diepte 1 van de FOBDT in de kwantificatiekern, ziet eruit als in figuur 4.2g, en heeft volgende eigenschappen:

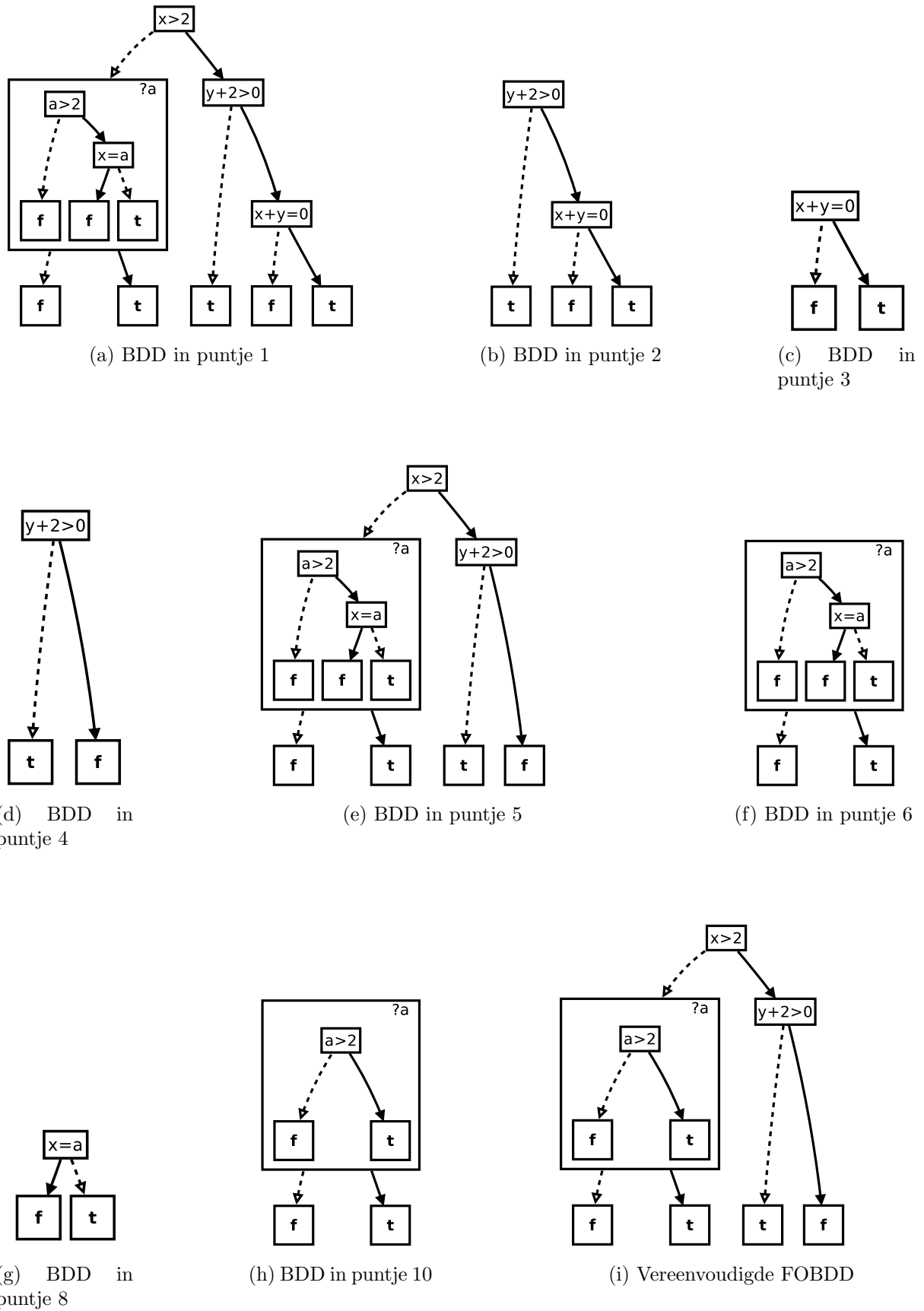
- $S = \{x \mid \neg x > 2 \wedge a > 2\}$
- De FOBDT is niet **t** of **f**
- De kern is geen kwantificatiekern
- De kern bevat een Presburger formule  $x > a$

Hier zien we dat de verzameling  $S'' = \{x, a \mid \neg x > 2 \wedge a > 2 \wedge x = a\}$  leeg is. We vervangen dus deze FOBDT door zijn falsetak, welke de triviale FOBDT **t** is, en keren terug naar diepte 0 van de FOBDT in de kwantificatiekern.

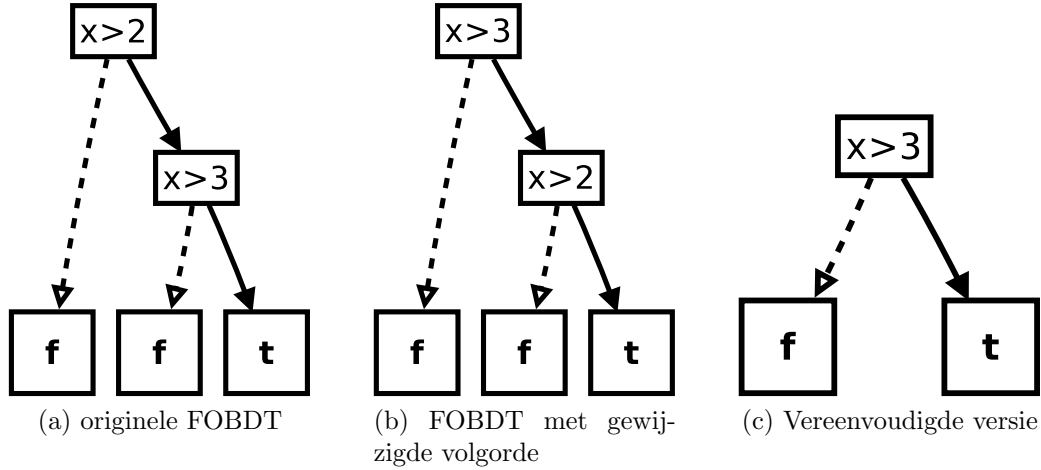
9. Deze FOBDT is dezelfde als in puntje 7, met vereenvoudigingen in een deelboom. We moeten nu de falsetak bezoeken, maar daar gaan we hier niet op in, aangezien dit de triviale FOBDT **f** is. We keren dus terug naar de FOBDT op diepte 1.
10. We hebben dus nu de vereenvoudigde versie van de FOBDT uit puntje 6, die eruit ziet als in figuur 4.2h. De takken van deze FOBDT zijn weerom triviale bomen, waar we niet op in gaan, we keren dus terug naar diepte 0, en krijgen de vereenvoudigde versie van de hele FOBDT, in figuur 4.2i.

## 4.6 Uitbreidingen

Het algoritme dat we geconstrueerd hebben in hoofdstuk 4.4, kan de problemen die we gesteld hebben aan het einde van het vorige hoofdstuk oplossen. Als een knoop Presburger redundant is, wordt de FOBDT vereenvoudigd, en wordt de redundante knoop



Figuur 4.2: FOBDTs



Figuur 4.3: Aritmetische redundantie bij bepaalde volgordes

weggehaald. We bekijken in dit hoofdstuk enkele kleine uitbreidingen op dit algoritme, die de werking nog verder kunnen verbeteren.

Voor een eerste uitbreiding bekijken we volgend voorbeeld: een FOBDT voor  $x > 2 \wedge x > 3$  (figuur 4.3). Op de FOBDT in figuur 4.3a zal ons algoritme niets kunnen vereenvoudigen, aangezien zowel de kern met formule  $x > 2$  als die met  $x > 3$  niet aritmetisch redundant is. Een kleine wijziging aan de volgorde van de kernen (figuur 4.3b), die niets aan de betekenis van de FOBDT wijzigt, zorgt echter ineens dat we wel een vereenvoudiging kunnen doorvoeren, aangezien de knoop met formule  $x > 2$  nu wel Presburger redundant is. Het blijkt dus dat we de kernen in verschillende volgordes gaan moeten plaatsen om echt te kunnen testen of er kernen zijn, die overbodig zijn in de formule die de FOBDT voorstelt. Gelukkig hebben we volgend resultaat, wat ons zegt dat we bij een FOBDT met  $n$  knopen toch niet alle  $n!$  volgordes zullen moeten proberen:

**Stelling 4.2.** *Zij  $B$  een FOBDT, die enkel Presburger formules bevat, met een kern  $K$  (die voorgesteld wordt door een formule  $\varphi$ ) waarvoor geldt dat we de kernen in  $B$  kunnen herordenen zodat er een voorkomen van  $K$  is dat Presburger redundant is in die herordende versie  $B'$  van  $B$ . Dan geldt voor elke herordende versie van  $B$  volgens een ordening waar  $K$  als laatste (dus helemaal beneden) staat dat  $K$  een voorkomen heeft dat Presburger redundant is.*

*Bewijs.* Zij  $B$  een FOBDT met een kern  $K$  die een voorkomen heeft dat Presburger redundant is. We bewijzen nu dat in elke herordening van  $B$  waarin  $K$  als laatste staat, er een voorkomen van  $K$  bestaat dat Presburger redundant is. Lemma 4.2.1 zegt ons nu dat het voldoende te bewijzen is dat er een herordende versie  $B'$  van  $B$  bestaat waarin  $K$  als laatste staat, en waarvoor geldt dat  $K$  een voorkomen heeft dat Presburger redundant is.

Noem daarom  $Q$  het pad naar het Presburger redundant voorkomen  $K'$  in  $B$ , en  $\alpha$  de formule die  $Q$  voorstelt. Omdat  $K'$  Presburger redundant is geldt dus dat  $\alpha \Rightarrow \varphi$  of  $\alpha \Rightarrow \neg\varphi$  waar is in elk model dat aan de Presburger axioma's voldoet. Als we nu  $K$



naar beneden schuiven (wisselen met de knoop die na  $K$  komt in de volgorde), dan is het duidelijk dat  $Q$  een deelpad is van elk pad naar elk voorkomen  $K''$  van  $K$  dat zich bevindt in de deelboom die juist  $K'$  als wortel had, waaruit dan volgt dat  $K''$  Presburger redundant is. We kunnen dus deze redenering blijven herhalen tot  $K'$  helemaal beneden staat, wat de stelling bewijst.  $\square$

**Lemma 4.2.1.** *Zij  $B$  een FOBDT, die enkel Presburger formules bevat, en zij  $K$  een kern die als laatste in de ordening van de kernen van  $B$  staat, en een Presburger redundant voorkomen heeft in  $B$ . Dan geldt voor elke herordende versie  $B'$  van  $B$  volgens een ordening waarin  $K$  nog steeds beneden staat, dat  $K$  nog steeds een Presburger redundant voorkomen heeft.*

*Bewijs.* Omdat elke herordening van kernen uitgevoerd kan worden door een samenstelling van wissels van twee opeenvolgende kernen, is het voldoende te bewijzen dat na het omwisselen van 2 kernen  $K_i$  en  $K_{i+1}$  die zich boven  $K$  bevinden,  $K$  nog steeds een Presburger redundant voorkomen heeft. Zij  $K'$  de knoop die Presburger redundant is,  $\varphi$  de formule die  $K$  voorstelt,  $P$  het pad naar  $K'$ , en  $\psi$  de formule die  $P$  voorstelt<sup>7</sup>. Omdat  $K'$  Presburger redundant is, weten we dat  $\psi \Rightarrow \varphi$  of  $\psi \Rightarrow \neg\varphi$  geldt. Veronderstel verder dat  $\psi \Rightarrow \varphi$ , het bewijs voor het geval dat  $\psi \Rightarrow \neg\varphi$  verloopt analoog. We onderscheiden 3 gevallen:

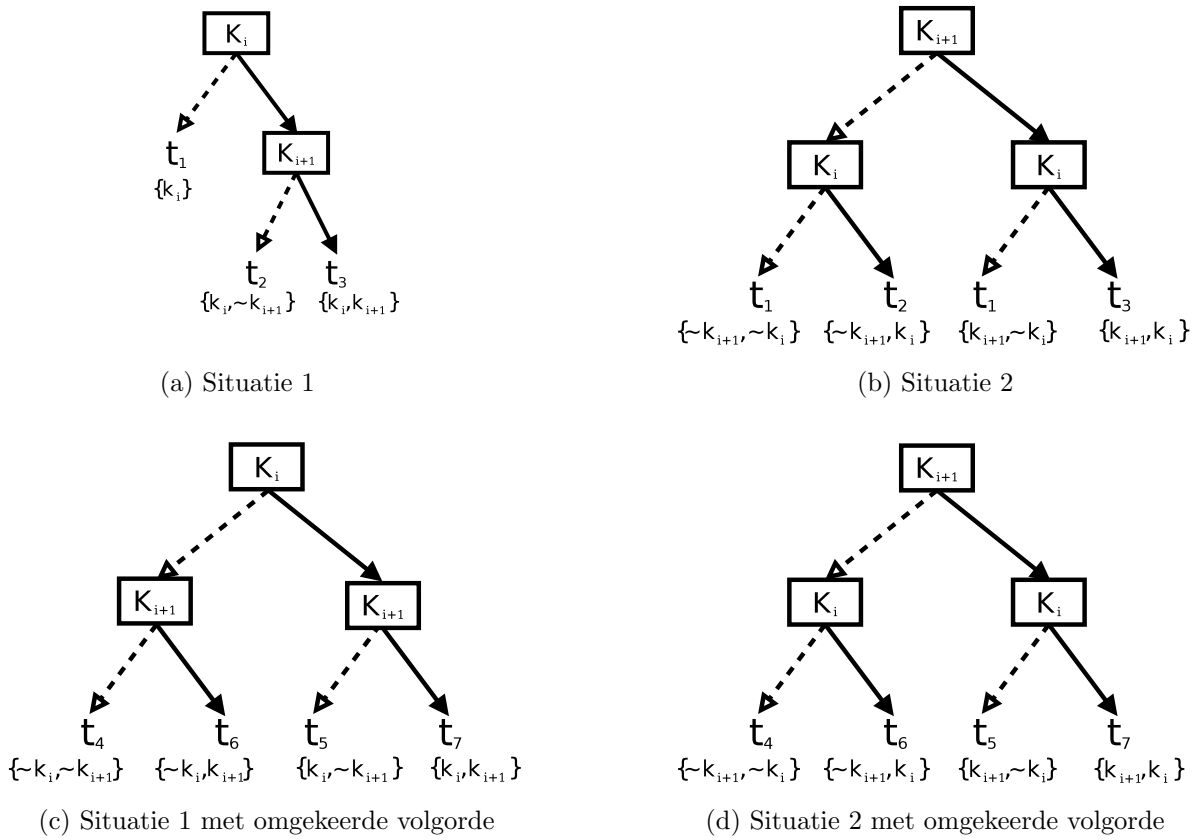
1. Als  $K_i$  en  $K_{i+1}$  geen van beide voorkomen in  $P$ , dan heeft het omwisselen van deze knopen ook geen enkele invloed op dit pad<sup>8</sup>. Dus is  $P$  nog altijd een pad in de herordende versie naar een voorkomen van  $K$ , dat dus Presburger redundant is.
2. Stel voor het tweede geval dat  $K_{i+1}$  voorkomt in  $P$ , maar  $K_i$  niet. Om de bewering voor dit geval te bewijzen merken we eerst op dat het voldoende is de bewering te bewijzen in het speciale geval dat  $K_{i+1}$  de wortel van  $B'$  is. Inderdaad, want analoog aan de redenering in puntje 1 weten we dat het pad  $P_1$  naar  $K_{i+1}$  in  $B'$  (waarvoor dus geldt dat  $P_1 \subset P$ ) niet wijzigt door de herordening, aangezien dit zowel  $K_i$  als  $K_{i+1}$  niet bevat. Omdat nu geldt dat  $K_i$  boven  $K_{i+1}$  in de volgorde staat geldt dat  $K_i$  niet voorkomt in  $B$ , en dat het omwisselen van de volgorde niets verandert aan  $B'$ .
3. Stel nu dat  $K_i$  wel voorkomt in  $P$  (al dan niet met  $K_{i+1}$ ), dan kunnen we analoog als in puntje 2 aannemen dat  $K_i$  de wortel van  $B'$  is. Stel dus dat  $K_i$  de wortel van  $B'$  is. Aangezien  $K_{i+1}$  vlak na  $K_i$  staat in de volgorde, is  $K_{i+1}$  dus ofwel wortel van beide takken, ofwel wortel van één tak, ofwel komt  $K_{i+1}$  niet voor. Als  $K_{i+1}$  niet voorkomt, kunnen we de redenering van puntje 2 herhalen om te vinden dat  $K$  nog steeds een Presburger redundant voorkomen heeft. Als  $K_{i+1}$  wel voorkomt, dan ziet  $B$  eruit als in figuur 4.4a, of als in figuur 4.4b. Van deze twee figuren zien we de herordende versies in respectievelijk figuur 4.4c en figuur 4.4d.

<sup>7</sup>Hiermee bedoelen we de conjunctie van alle formules in de padformuleverzameling van  $P$  (zie hoofdstuk 4.1).

<sup>8</sup>De reden hiervoor is dat het wisselen van twee knopen enkel de verwijzingen vanuit deze twee knopen beïnvloedt([Rud93], )

Aangezien de deelbomen  $t_i$  ( $i=1..4$ ) niet beïnvloed worden door de herordening, moeten we enkel nog aantonen dat de nieuwe padformuleverzamelingen voor de wortels van de deelbomen  $t_i$  ( $i=1..4$ ) bevat zijn in de oude. Want dan geldt dat de conjunctie van de formules in de nieuwe padformuleverzameling, de conjunctie van de formules in de oude padformuleverzameling impliceert, en bijgevolg ook  $\varphi$  impliceert. De padformuleverzamelingen zien we in figuur 4.4, waar  $k_i$  en  $k_{i+1}$  de formules zijn die  $K_i$  respectievelijk  $K_{i+1}$  voorstellen. Het is duidelijk uit deze figuur dat de Presburger redundantie bewaard blijft.

□



Figuur 4.4: FOBBDTs bij bewijs van Stelling 4.2

Stelling 4.2 is een sterk resultaat, zoals we kunnen zien aan dit gevolg, wat in principe gewoon de omgekeerde formulering is van de stelling.

**Gevolg 4.2.1.** *Zij  $B$  een FOBBDT, die enkel Presburger formules bevat, en zij  $K$  een kern die als laatste in de ordening van de kernen van  $B$  staat. Als er geen enkel voorkomen van  $K$  bestaat Presburger redundant is, dan bestaat er geen enkele herordening van  $B$  waarin  $K$  wel een Presburger redundant voorkomen heeft.*

Stelling 4.2 is ook perfect veralgemeenbaar tot FOBBDTs die niet enkel Presburger formules bevatten. Ons algoritme negeert de niet-Presburger formules in een dergelijke FOBBDT

toch, waardoor de FOBDT eigenlijk behandeld wordt alsof hij enkel Presburger formules zou bevatten. Hieruit volgt dus dat we voor elke FOBDT  $B$ , maximaal  $n$  volgordes moeten testen, met  $n$  het aantal Presburger kernen.

We komen dan uiteindelijk tot volgend algoritme (algoritme 4.3, waarin elke kern één keer naar beneden geschoven wordt, waarna het oorspronkelijke algoritme wordt uitgevoerd). Hierbij moet het oorspronkelijke algoritme eventueel enkel worden aangepast op regel 4. We kunnen daar  $\text{simpBDT1}(\text{FOBDT in } B.\text{kern})$  vervangen door  $\text{simpBDT2}(\text{BDD in } B.\text{kern})$ , als we binnen die kwantificatie ook verschillende volgordes willen testen.

---

**Algoritme 4.3:**  $\text{simpBDT2}$ : aritmetische vereenvoudiging van FOBDTs, onafhankelijk van de volgorde van de kernen

---

**Input:** Een FOBDT  $B$

**Output:**  $B$  wordt vereenvoudigd

```

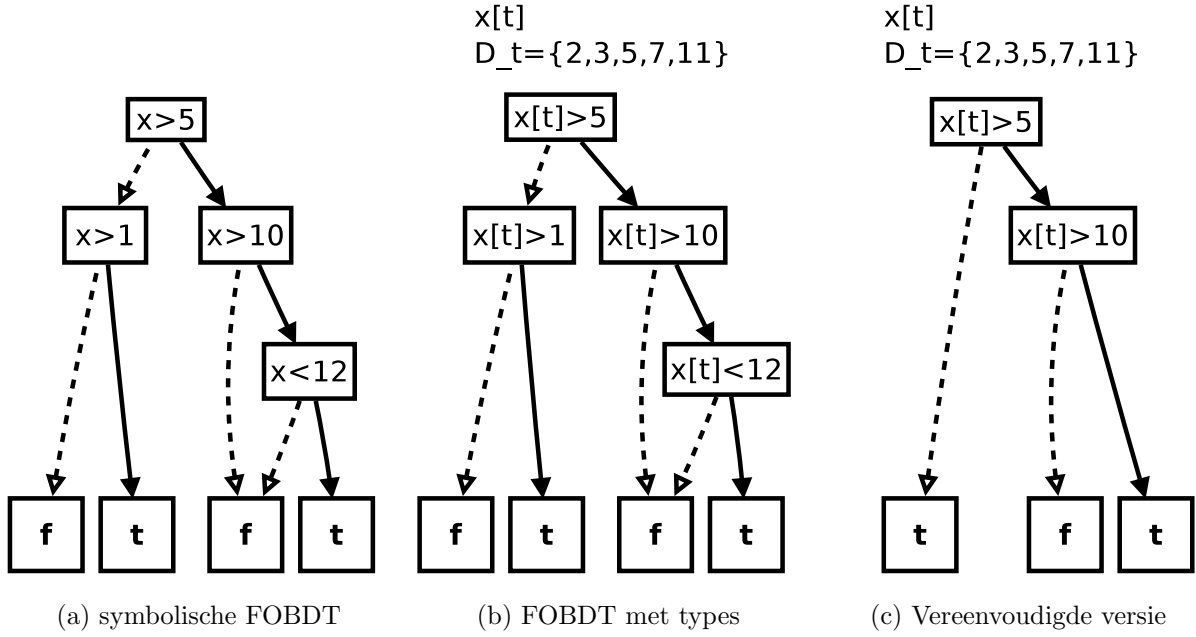
1  $B' = B$ ;
2 for  $K \text{ kern} \in B$  do
3   if  $K \in B'$  then
4     Zet  $K$  op laatste plaats;
5      $B' = \text{simpBDT1}(B')$ ;
6   end
7 end
8 return  $B'$ ;
```

---

Het algoritme tot nu toe is perfect bruikbaar voor symbolische FOBDTs in een uitbreiding van eerste ordeloga waar aritmetiek inzit. Dit algoritme werd ontwikkeld met als doel gebruikt te worden in het IDP-systeem, waarin getypeerde eerste-ordeloga gebruikt wordt. Deze typering is vooral interessant in de context van een grounding, waar er al een domein voor de types van de variabelen in de FOBDT gekend is. In het geval dat het type  $t$  van een variabele  $x$  een subtype is van de gehele getallen, dan kunnen we de voorwaarde  $\text{type}(x) = t$  vaak voorstellen door een Presburger formule, en dus deze voorwaarde gebruiken in ons algoritme.

Stel als triviaal voorbeeld dat we een FOBDT hebben voor een formule  $\exists x(x < 0)$ , dan kunnen we hier niets aan vereenvoudigen, want zolang we niets weten over  $x$  kunnen we ook niet besluiten of  $x < 0$  kan zijn. Maar stel dat de formule er zo uit ziet:  $\exists x[N](x < 0)$  en dat we weten dat het domein van  $N = \mathbb{N}$ , dan kunnen we dit wel vereenvoudigen tot **f**. Voor een groter voorbeeld kijken we naar figuur 4.5. In figuur 4.5a zien we een FOBDT voor de formule  $(\neg(x > 5) \wedge (x > 1)) \vee ((x > 10) \wedge (x < 12))$ , waar geen enkele Presburger redundante knoop in zit. Stel echter dat we deze FOBDT bekijken in de context van een model expansie waarbij  $x$  type  $t$  heeft, en  $t$  een domein  $D_t = \{2, 3, 5, 7, 11\}$  heeft, dan krijgen we de FOBDT in figuur 4.5b. Hier zien we wel verschillende Presburger redundante knopen, bijvoorbeeld de knoop die  $x < 12$  bevat (want als  $x > 10$ , moet  $x = 11$  en dus kleiner dan 12 zijn). De vereenvoudigde FOBDT zien we tenslotte in figuur 4.5c.

Het is duidelijk dat het gebruik van deze voorwaarden, die we uit het type van onze



Figuur 4.5: Presburger redundantie in getypeerde FOBDTs

variabelen kunnen halen (en verder als typevoorwaarden gaan benoemen), in sommige gevallen voor verdere vereenvoudiging van een FOBDD kan zorgen. Elke keer we dus in algoritme 4.5 een variabele tegenkomen die nog niet in  $S$  zit, voegen we hiervan het type toe. Het toevoegen zelf gebeurt met algoritme 4.4. Als het domein voor het type  $t$  gegeven wordt door een interval, dan maken we hier gebruik van, en dan moeten we dus niet alle elementen apart toevoegen<sup>9</sup>.

## 4.7 Implementatie

Om de overzichtelijkheid te bewaren hebben we in de vorige hoofdstukken de algoritmes op hoog niveau beschreven. Deze algoritmes zijn natuurlijk ook geïmplementeerd in het IDP-systeem, en in dit hoofdstuk beschrijven we enkele aspecten van deze implementatie en gaan we wat dieper in op sommige algoritmes. We beschrijven eerst het algoritme dat we nodig hadden in hoofdstuk 4.3.3: het algoritme om nieuwe formules toe te voegen aan een Omega+ verzameling. Daarna bekijken we nog enkele andere details waar we niet eerder op ingegaan zijn, onder meer hoe we ons algoritme moeten aanpassen om te werken met FOBDDs in plaats van FOBDTs (hoofdstuk 4.7.2), en hoe we het moeten aanpassen om te werken met de Bruijn-indices, want deze worden ook gebruikt in IDP (hoofdstuk 4.7.3). Ten slotte bekijken we nog enkele aspecten van de eigenlijke implementatie, en het programmeren (hoofdstuk 4.7.4).

<sup>9</sup>In de praktijk bleek dat het apart toevoegen van alle domeinelementen voor grote domeinen enorm tijdsintensief was. Omdat de typevoorwaarden niet volledig moeten zijn, is er besloten om ook bij opsommingen de formule op regel 4 toe te voegen.

---

**Algoritme 4.4:**  $\text{voegTypevoorwaardenToe}(S, x)$ : Het toevoegen van voorwaarden die volgen uit het type van een variabele.

---

**Input:** Omega+ verzameling  $S$ , Een variabele  $x$

**Output:** Aangepaste verzameling  $S'$

---

```

1  $t = x.type();$ 
2 if  $t$  is subtype van  $int$  then
3    $D = t.domain();$ 
4   if  $D$  is een interval then
5      $\psi := D.min() \leq x \leq D.max();$ 
6   else
7     for  $d \in D$  do
8        $\psi_d := x = d;$ 
9     end
10     $\psi := \bigvee_{d \in x.type()} \psi_d;$ 
11  return  $BreidUitEnVoegVoorwToe(S, \psi);$ 
12
```

---

#### 4.7.1 Het toevoegen van formules aan een Omega+ verzameling

In hoofdstuk 4.3.3 werd beschreven hoe we kunnen beslissen of een bepaalde knoop  $K$  die voorgesteld wordt door formule  $\varphi$  Presburger redundant is in een FOBDT  $B$ . We houden een Omega+ verzameling  $S$  bij, waarin op elk moment alle  $n$ -tupels zitten die voldoen aan de formule  $\psi$  (met  $n$  vrije variabelen) die het pad naar  $K$  voorstelt. Het testen of een knoop redundant is, doen we dan door twee nieuwe verzamelingen aan te maken. We maken een verzameling  $S_0$  door  $\varphi$  toe te voegen aan  $S$ , en een verzameling  $S_1$  door  $\neg\varphi$  toe te voegen aan  $S$ . Concreet betekent dit dus dat we een algoritme nodig hebben dat gegeven een Omega+ verzameling

$$S = \{x_1, \dots, x_n | \psi[x_1, \dots, x_n]\}$$

van dimensie  $n$  en een formule  $\varphi[x_k, \dots, x_n, x_{n+1}, \dots, x_m]$  met  $m - k$  vrije variabelen (waarbij we met  $x_k, \dots, x_n$  de variabelen bedoelen die al in  $S$  voorkomen, en  $x_{n+1}, \dots, x_m$  de variabelen bedoelen die nog niet in  $S$  voorkomen<sup>10</sup>) een Omega+ verzameling  $S'$  kan construeren van dimensie  $m$  zodat

$$S' = \{x_1, \dots, x_k, \dots, x_n, \dots, x_m | \psi[x_1, \dots, x_n]\}$$

Algoritme 4.5 voegt een gegeven Presburger formule  $\varphi$  toe aan een gegeven Omega+ Verzameling  $S$ . Stel dat  $S$  een  $n$ -dimensionale verzameling is, waaraan we een Presburger formule  $\varphi$  willen toevoegen met  $m - k$  vrije variabelen (zoals hierboven). Eerst wordt er gekeken of  $\varphi$  vrije variabelen bevat die nog niet in  $S$  voorkomen. Als dit het geval is wordt de dimensie van  $S$  uitgebreid, en worden de namen van de nieuwe variabelen toegewezen voor de nieuwe dimensies. Hierna maken we een nieuwe Omega+ verzameling

---

<sup>10</sup>Als  $k = n + 1$ , dan bedoelen we dat  $\varphi$  geen vrije variabelen heeft die in  $S$  voorkomen, en als  $m = n$ , dan bedoelen we dat  $\varphi$  geen nieuwe vrije variabelen heeft.

$T$  met dezelfde variabelen en met als voorwaarde de formule die we willen toevoegen aan  $S$ . We nemen dan uiteindelijk de doorsnede van  $S$  en  $T$ , aangezien dit de verzameling is van alle  $m$ -tupels die zowel aan de voorwaarden van  $S$  als aan  $\varphi$  voldoen (waarbij  $m$  het aantal vrije variabelen in  $\varphi$  is, die nog niet in  $S$  voorkwamen).

---

**Algoritme 4.5:** BreidUitEnVoegVoorwToe( $S, \varphi$ ): Breidt Omega+ verzameling  $S$  uit met voorwaarde  $\varphi$

---

**Input:** Omega+ verzameling  $S$ , formule  $\varphi$

**Output:** Aangepaste verzameling  $T$

---

```

1  $\Gamma$  = vrije variabelen van  $\varphi$ ;
2  $\Sigma$  = variabelen van  $S$ ;
3 for  $x \in \Gamma \setminus \Sigma$  do
4   | Voeg variabele toe aan  $S$  ;
5   | voegTypevoorwaardenToe( $S, x$ );
6 end
7  $T$  = new OmegaSet( $S.variables, \varphi$ ) ;      // OmegaSet(variabelen, formule)
8 return  $S \cap T$ ;
```

---

Stel ter illustratie dat  $S$  een verzameling is van dimensie 2, en dat we er een formule  $\varphi$  aan willen toevoegen met één nieuwe variabele. We kunnen  $S$  voorstellen als een veelhoek in het vlak, waarbij de zijden van deze veelhoek gegeven worden door de voorwaarden van de formules in  $S$ . Omdat  $\varphi$  een vrije variabele bevat die  $S$  niet bevat, moeten we  $S$  met één dimensie uitbreiden. De uitbreiding  $S'$  is dan een ruimtelijke figuur, die we ons kunnen voorstellen als een cilinder van oneindige lengte met grondvlak  $S$ . De verzameling  $T$  is dan een andere ruimtelijke figuur, waarbij de grenzen gegeven worden door  $\varphi$ . De ruimtelijke figuur die dan alle punten bevat die voldoen aan  $\psi \wedge \varphi$ , bevinden zich dan in de doorsnede van  $S'$  en  $T$ .

### 4.7.2 Aanpassingen voor FOBDDs

Het werken met FOBDDs in plaats van met FOBDDTs brengt eigenlijk niet zoveel veranderingen met zich mee. Algoritme 4.2 moet een klein beetje aangepast worden, we mogen namelijk niet meer zomaar kernen en deelbomen vervangen, aangezien het kan zijn dat deze nog ergens anders gebruikt worden. Het aangepaste algoritme staat in algoritme 4.6. We zien dat we bij het doorlopen van een FOBDD de kern, true- en falsetak niet overall onmiddellijk vervangen, maar we houden de nieuwe versies bij tot op het einde van het algoritme. Daar wordt een nieuwe FOBDD gevraagd, met de nieuwe kern en de twee nieuwe takken. De methode die deze FOBDD aanmaakt zorgt dat de FOBDD gereduceerd is, en kent alle FOBDDs die al ooit aangemaakt zijn. Er wordt dan eerst opgezocht of er geen FOBDD bestaat met deze kern en takken. Enkel als dit niet het geval is, wordt er een nieuwe FOBDD aangemaakt, anders wordt er een verwijzing naar die al bestaande gelijke FOBDD teruggegeven.

---

**Algoritme 4.6:** simpFOBDD: aritmetische vereenvoudiging van FOBDDS

---

**Input:** Een FOBDD  $B$ **Interne variabelen:** Omega+ verzameling  $S$ **Output:** Aritmetisch vereenvoudigde versie van  $B$ 

```

1 if  $B = \text{true}$  of  $B = \text{false}$  then
2   | return  $B$ ;
3 else
4   |  $S_{\text{backup}} = S$  ; // Maak een back-up van  $S$ 
5   | if  $B.\text{kern}$  is een kwantificatiekern then
6     |  $v = \text{gekwantificeerde variabele in } B.\text{kern}$ ;
7     |  $B_Q = \text{simpFOBDD}(\text{FOBDD in } B.\text{kern})$ ;
8     |  $\text{kern2} = \text{new QKern}(v, B_Q)$ ;
9     |  $S = S_{\text{backup}}$  ; // Herstel back-up
10  |  $\varphi = \text{formule die kern van } B \text{ voorstelt}$ ;
11  | if  $\varphi$  is een Presburger formule then
12    |  $S' = \text{BreidUitEnVoegVoorwToe}(S, \neg\varphi)$ ;
13    | if  $S' = \text{leeg}$  then
14      | return  $\text{simpFOBDD}(B.\text{truetak})$ ;
15    |  $S'' = \text{BreidUitEnVoegVoorwToe}(S, \varphi)$ ;
16    | if  $S'' = \text{leeg}$  then
17      | return  $\text{simpFOBDD}(B.\text{falsetak})$ ;
18  |
19  | if  $\varphi$  is een Presburger formule then
20    |  $S = S.\text{BreidUitEnVoegVoorwToe}(\varphi)$ ;
21  |  $\text{truetak2} = \text{simpFOBDD}(B.\text{truetak})$ ;
22  |  $S = S_{\text{backup}}$  ; // Herstel back-up
23  | if  $\varphi$  is een Presburger formule then
24    |  $S = S.\text{BreidUitEnVoegVoorwToe}(\neg\varphi)$ ;
25  |  $\text{falsetak2} = \text{simpFOBDD}(B.\text{falsetak})$ ;
26  |  $S = S_{\text{backup}}$  ; // Herstel back-up
27  |  $B_2 = \text{geeffOBDD}(\text{kern2}, \text{truetak2}, \text{falsetak2})$ ;
28  | return  $B_2$ ;
29

```

---

### 4.7.3 Aanpassingen voor de Bruijn-indices

Zoals aangehaald in hoofdstuk 3.2.3, worden in IDP de Bruijn-indices gebruikt om gebonden variabelen voor te stellen. Als we in ons algoritme dus een kwantificatiekern vereenvoudigen door de FOBDD in de kern te vereenvoudigen, staan hier indices in, die verwijzen naar een kwantor die er niet meer is. Dit probleem werd opgelost door eerst door de hele FOBDD te lopen en elk voorkomen van een de Bruijn-index die bij de kwantor hoort te vervangen door een nieuwe variabele  $a$ . Dit doen we door eerst elke index  $< 0 >$  te vervangen, en als we een kwantificatiekern binnengaan, daarbinnen elke index  $< 1 >$  te vervangen, en als we daarbinnen gaan, elke index  $< 2 >$  te vervangen, enz... Na dit substitueren van indices hebben we een FOBDD waar de de Bruijn-index die bij de weggenomen kwantor hoorde niet meer in voorkomt. Als we dit altijd zo doen, dan zitten er nooit de Bruijn-indices in de FOBDD die we moeten vereenvoudigen, en kan ons algoritme deze FOBDD vereenvoudigen. Na het vereenvoudigen doen we weer het omgekeerde, we vervangen  $a$  door  $< 0 >$  zolang we geen kwantor zijn tegengekomen, en als we een kwantificatiekern ingaan verhogen we de index waarmee we  $a$  vervangen, zodat de verwijzingen naar de kwantoren altijd correct blijven.

### 4.7.4 Het vereenvoudigingsalgoritme

Algoritme 4.6 verschilt met de werkelijke implementatie op nog een paar aspecten. Het belangrijkste verschil, en het enige dat we hier gaan bespreken, is dat er in de implementatie geen recursief algoritme gebruikt wordt, zoals algoritme 4.2 wel is. De reden hiervoor is dat ons algoritme door zijn vorm heel slecht uitbreidbaar is. Ons algoritme behandelt namelijk enkel FOBDDs in (getypeerde) eerste-orde logica, terwijl er in het IDP-systeem bijvoorbeeld ook aggregaten voorkomen, die in FOBDDs worden weergegeven door een nieuw soort kern: een aggregaatkern. Zelfs als ons algoritme daar niets mee wilt doen, moet toch met het bestaan ervan rekening gehouden worden, en moet er dus ergens een conditie bijkomen die stelt dat de kern in de FOBDD geen aggregaatkern mag zijn. Bovendien geldt dat als we het algoritme recursief laten werken, dat we elke keer dat onderscheid moeten maken tussen de verschillende soorten kernen, en daarna moeten typecasten naar het juiste type om bijvoorbeeld de FOBDD in de kwantificatiekern te kunnen opvragen.

In de implementatie willen we dit door overerving laten werken, dus verschillende methodes `simpFOBDD(BDD)`, `simpFOBDD(kwantificatiekern)`, ... die elkaar oproepen. De meest voor de hand liggende manier hiervoor, is het toevoegen van een `simpFOBDD` methode in elke klasse van objecten die we willen vereenvoudigen. Het nadeel hiervan is dat dit heel erg onoverzichtelijk wordt als we veel algoritmes hebben die iets moeten wijzigen aan of weten over FOBDDs (wat in het IDP-systeem zeker het geval is). Om dit probleem op te lossen, wordt er in de implementatie van het IDP-systeem met het visitor ontwerp-patroon gewerkt ([GHJV94]). Door één klasse `FOBDDVisitor` te creëren, en alle klassen die een FOBDD moeten doorlopen hiervan te laten overerven, kunnen we alle standaardgevallen in deze ouder-klasse opvangen. We maken nu in elke klasse die een FOBDD, of een deel daarvan (bv. kern) voorstelt een methode `accept(Visitor v)`, waarin simpelweg staat `v.visit(this)`. Als we nu in ons algoritme een kern  $K$  tegenkomen, dan moeten we



niet uitzoeken wat voor kern het is, we geven gewoon de opdracht  $K.accept(this)$ . Dan wordt door overerving bepaald wat voor soort  $K$  is, en vanuit  $K$  wordt dan de opdracht  $v.visit(K)$  gegeven, waarmee door overerving en overloading bepaald kan worden welke methode er moet worden uitgevoerd op  $K$ . Deze programmeertechniek wordt “Double Dispatch” genoemd.

Algoritme 4.2 wordt dus in de praktijk opgesplitst in 3 algoritmes, namelijk één voor FOBDDs, één voor atomische kernen en één voor kwantificatiekernen.

## 4.8 Resultaten bij Experimenten

De probleemstelling die we geformuleerd hebben aan het einde van hoofdstuk 3, hebben we zo goed als mogelijk kunnen oplossen met ons algoritme. Het algoritme kan alle Presburger redundantie uit elke FOBDD halen. De hoofdreden voor het vereenvoudigen van FOBDDs was het verhogen van de efficiëntie van het IDP-systeem. In dit hoofdstuk onderzoeken we kort of dat gelukt is. We testen eerst verschillende modelexpansieproblemen<sup>11</sup>, en bekijken of deze al dan niet sneller opgelost worden. De resultaten van deze testen zijn gemengd. Bij het uitvoeren van grote modelexpansies zijn er problemen waar het vereenvoudigen van FOBDDs de uitvoering van het modelexpansie-algoritme versnelt, maar er zijn ook problemen waar het modelexpansie-algoritme hier net door vertraagd wordt. We bekijken in dit hoofdstuk ook de mogelijke oorzaken van deze matige resultaten en we kijken ook eens wat we kunnen doen om dit kunnen verbeteren.

De resultaten in dit hoofdstuk werden bekomen door alle testen 20 keer uit te voeren waarna we de gemiddelde uitvoeringstijd hebben berekend. Dit hebben we gedaan om statistisch betrouwbare resultaten te bekomen. We hebben bij het uitvoeren ook veel testen gevonden waar we spreken van uitvoeringstijden tot zelfs kleiner dan 5 honderdsten van een seconde. Het probleem met deze experimenten is dat het hier erg moeilijk wordt om te testen of ons algoritme zorgt voor het versnellen van de uitvoeringstijd, aangezien de meeste tijd niet gaat naar het uitvoeren van het algoritme, maar wel naar andere zaken zoals het opstarten van IDP. Die overhead is bij de meeste grote problemen te verwaarlozen, maar als we spreken over problemen met een uitvoeringstijd van 0.05 seconden, dan kunnen we dit niet verwaarlozen. Deze kleine problemen werden dus ook uit de resultaten gehaald. De experimenten werden uitgevoerd op een computer met volgende specificaties:

- Processor: Intel i5-2410M Dual-Core processor 2.3 GHz
- RAM: 3.8Gb Ram-geheugen
- Besturingssysteem: Ubuntu 11.04 met kernel 2.6.38-13

We bekijken eerst de resultaten bij het uitvoeren van een hele verzameling testen, zowel met als zonder het vereenvoudigen van aritmetische FOBDDs. Deze resultaten zijn te

---

<sup>11</sup>De testen zijn allemaal te vinden op <http://www.student.kuleuven.be/~s0193486/Thesis/>, en bevatten het vocabularium, de structuur en de theorie waarvoor de modelexpansie werd uitgevoerd.

vinden in tabel 4.1. De resultaten gaan alle richtingen uit, sommige problemen kunnen we sneller oplossen met ons algoritme, voor sommige maakt het weinig uit, maar er zijn ook problemen waar de oplossing voor vertraagt. De verbetering van de efficiëntie van het

Naam test	Met vereenv.	Zonder vereenv.	Verbetering	Relatieve verbetering
<i>Bin Packing 1</i>	2.01	2.54	0.53	26.5%
<i>Bin Packing 2</i>	1.54	2.62	1.08	41.2%
<i>Blocked N-Queens</i>	2.00	1.99	−0.01	−0.5%
<i>Hamiltonian Circuit</i>	1.09	1.09	0	0%
<i>Hamiltonian Path</i>	23.00	22.72	−0.27	−1.2%
<i>Hitori</i>	0.33	0.25	−0.08	−23.6%
<i>Knights</i>	1.50	1.27	−0.22	−14.8%
<i>Light</i>	4.10	3.30	−1.20	−19.6%
<i>Mirror</i>	2.28	2.49	0.21	9.3%
<i>Missionaries</i>	2.00	1.99	−0.01	−0.5%
<i>N-Queens</i>	16.21	9.03	−7.18	−44.3%
<i>Pigeonhole</i>	11.00	12.00	1.00	9%
<i>Schildpad</i>	1.39	1.58	0.19	13.5%
<i>Social Golfer</i>	4.10	4.26	0.16	3.9%
<i>Sokoban</i>	13.13	13.92	0.79	6%
<i>Sudoku</i>	0.50	0.68	0.18	26.5%
<i>Tarski puzzel</i>	6.02	6.04	0.02	0.3%
<i>Treinplanning</i>	5.90	6.00	0.10	1.5%
<i>Waterbucket</i>	25.28	29.36	4.08	16.1%
<i>Zeeslag</i>	3.88	3.44	−0.44	−11.4%

Tabel 4.1: De tijdsduur bij het uitvoeren van verschillende testen, met en zonder het vereenvoudigingsalgoritme

Bin Packing probleem<sup>12</sup>, is wel erg goed. Het eerste Bin Packing probleem is een klein probleem, het gaat over 3 vierkanten en een rechthoek van grootte  $4 \times 6$ . Als het probleem groter wordt, zoals Bin Packing probleem 2 (9 vierkanten en een  $13 \times 7$  domein) zien we dat ons algoritme ook meer kan verbeteren, en dus het grounden verder versnelt.

We zien echter in tabel 4.1 ook verschillende resultaten die minder positief waren, zoals bijvoorbeeld het N-Queens probleem. Dit betekent echter nog niet dat het ontwikkelde algoritme niet zinvol is. De plaats in de grounding waar het vereenvoudigen van FOBDDs erg veel kan opbrengen, is tijdens het gebruik van grenzen bij het grounden. Zoals we hebben bekeken in hoofdstuk 3.2.1 geldt dat er een afweging gemaakt moet worden over de scherpste van de gebruikte grenzen, wat zich vertaalt naar grootte van de gebruikte FOBDDs. Als we deze grenzen scherper maken, zullen de grenzen vaker gebruikt kunnen worden, maar wordt de query die de grenzen gebruikt ook duurder. Wat de ideale grootte van FOBDDs hierin is, is nog niet volledig gekend, dit wordt namelijk nog volop onderzocht. Wat we echter wel weten, is dat scherpere grenzen meer voordeel leveren bij

<sup>12</sup>Gegeven een aantal vierkanten van verschillende grootte, en een grote rechthoek, moeten we de vierkanten zo schikken dat ze allemaal in de rechthoek passen.

grote problemen en meestal vertragen bij kleine problemen. De problemen die wij getest hebben, zijn over het algemeen kleinere problemen en er werd een maximale diepte van 8 knopen ingesteld (hiermee bedoelen we dat elk pad maximaal 8 knopen kan bevatten).

Van ons algoritme verwachten we echter de beste resultaten bij grote FOBDDs. Dit komt enerzijds omdat daar het meeste aritmetische redundantie inzit, en omdat het verwijderen van een deelboom in een grote FOBDD, een grote deelFOBDD kan zijn, wat dus grote gevolgen kan hebben. We hebben dan ook getest of ons algoritme in de praktijk ook beter presteert als we grotere FOBDDs gebruiken. We hebben dus de maximale diepte van de FOBDDs verdubbeld, en de test nog eens opnieuw uitgevoerd. De resultaten van dit experiment zien we in tabel 4.2. Zoals we verwacht hadden, vertragen bijna alle problemen ten opzichte van 4.1, omdat het kleine problemen zijn die we testen, maar zien we wel dat ons algoritme voor veel grotere verschillen zorgt.

Naam test	Met vereenv.	Zonder vereenv.	Verbetering	Relatieve verbetering
<i>Bin Packing 1</i>	4,99	20,01	15,02	75,1%
<i>Bin Packing 2</i>	4,03	18,02	13,99	77,6%
<i>Blocked N-Queens</i>	1,99	2,59	0,60	23,2%
<i>Hamiltonian Circuit</i>	1,10	1,10	0,00	0,2%
<i>Hamiltonian Path</i>	24,00	23,00	-1,00	-4,3%
<i>Hitori</i>	0,79	0,80	0,01	1,0%
<i>Knights</i>	1,47	2,60	1,13	43,3%
<i>Light</i>	3,30	3,30	0,00	0,2%
<i>Mirror</i>	2,63	2,61	-0,03	-1,0%
<i>Missionaries</i>	3,29	2,59	-0,71	-27,3%
<i>N-Queens</i>	17,03	16,03	-1,00	-6,3%
<i>Pigeonhole</i>	12,99	13,00	0,01	0,1%
<i>Schildpad</i>	11,99	17,99	6,00	33,3%
<i>Social Golfer</i>	19,00	5,99	-13,00	-217,0%
<i>Sokoban</i>	132,02	192,03	60,01	31,3%
<i>Sudoku</i>	0,49	0,60	0,10	17,3%
<i>Tarski puzzel</i>	12,02	12,02	0,00	0,0%
<i>Treinplanning</i>	7,99	7,99	0,00	0,1%
<i>Waterbucket</i>	88,01	97,00	8,99	9,3%
<i>Zeeslag</i>	53,00	65,01	12,01	18,5%

Tabel 4.2: De tijdsduur bij het uitvoeren van verschillende testen met grotere FOBDDs, met en zonder het vereenvoudigingsalgoritme

# Hoofdstuk 5

## Besluit

Ter afsluiting van deze thesis overlopen we nog eens alles wat we in dit werk gezien en bereikt hebben. De bedoeling van deze thesis was een algoritme te ontwikkelen voor gebruik in IDP, dat aritmetische redundantie uit formules kan halen, waardoor het grounden in IDP sneller en efficiënter verloopt. Deze doelstelling is deels geslaagd. Het algoritme is geconstrueerd, en voldoet aan de verwachtingen: het kan allerlei aritmetische redundanties in een eerste-ordeformule opsporen, en deze elimineren. Het gevolg van deze verbetering in de praktijk op de efficiëntie van het IDP-systeem is nog niet in alle gevallen zo positief als we gehoopt hadden.

### 5.1 Theorie

In hoofdstuk 2 hebben we BDDs ingevoerd. Dit hebben we gedaan door eerst BDTs in te voeren, een normaalvorm met een binaire boomstructuur voor propositionele formules. We hebben dan enkele opmerkingen gegeven over de nadelen van BDTs, en met deze opmerkingen zijn we tot het concept van BDDs gekomen, een canonieke normaalvorm voor propositionele logica. Omdat deze normaalvorm zo interessant is voor propositionele formules, hebben we de ideeën hiervan veralgemeend naar eerste-orde logica, en kwamen we tot FOBDDs, eerste-orde BDDs. Deze normaalvorm voor eerste-orde logica was dan wel niet canoniek, maar heeft toch een aantal interessante eigenschappen, die we bekeken hebben.

In het volgende hoofdstuk van deze thesis hebben we het IDP-systeem, het systeem dat we wilden verbeteren, van naderbij bekeken. We zijn hier dieper ingegaan op de werking van de modelexpansie, en hoe hierbij een zo hoog mogelijke efficiëntie bereikt kan worden. Specifiek voor het grounden overliepen we een algoritme uit de literatuur waarbij FOBDDs gebruikt worden om grenzen op te stellen, waarmee we de grounding kunnen reduceren. Met het introduceren van die FOBDDs in de context van modelexpansie kwamen we tot onze probleemstelling. Als we willen dat het groundingsalgoritme efficiënt is, is het belangrijk dat deze FOBDDs ook efficiënt zijn. Eén van de problemen in het algoritme was dat aritmetische redundantie in een FOBDD het systeem in sommige gevallen enorm

kon vertragen. We hebben enkele voorbeelden bekeken van gevallen waarin dit werkelijk een probleem vormde.

De hoop om aritmetische redundantie volledig uit een FOBDD te kunnen verwijderen hebben we snel laten varen, aangezien er theoretische resultaten zijn, die ons vertellen dat dit niet mogelijk is. In plaats van ons doel daarmee op te geven, hebben we getracht een manier te zoeken om toch zo veel mogelijk redundantie uit een FOBDD te halen. We hebben daarbij gezocht naar een eerste-ordetheorie voor de gehele getallen, die beslisbaar is, en die we konden gebruiken om deze redundantie op te sporen. Bij deze zoektocht zijn we terecht gekomen bij Presburger aritmetiek, in hoofdstuk 4.3.1 hebben we deze theorie formeel gespecificeerd, en bewezen dat hij beslisbaar is.

## 5.2 Algoritme

We hebben in deze thesis een algoritme geconstrueerd dat gegeven een FOBDD, redundante Presburger formules kan opsporen, en deze kan verwijderen. Dit algoritme wordt geconstrueerd in hoofdstuk 4. Het maakt gebruik van de boomstructuur van de FOBDD, waarin heel makkelijk kan gezien worden of een deelformule redundant is. We zoeken in de FOBDD naar takken die onbereikbaar zijn, en we snoeien deze takken weg.

Er werd nog onderzocht of we dit algoritme met enkele kleine uitbreidingen nog bruikbaar zouden kunnen maken in IDP, en er werden twee uitbreidingen geformuleerd. Een eerste uitbreiding gaat over verschillende FOBDDs die dezelfde formule voorstellen, en waarin we andere redundanties kunnen ontdekken. De tweede uitbreiding gaat over het specifiek gebruik van FOBDDs in de context van modelexpansie in IDP, waarin een uitbreiding van eerste-ordelogica gebruikt wordt ( $\text{FO}(\cdot)$ ).

## 5.3 Implementatie

Dit algoritme, en zijn uitbreidingen, werden ook geïmplementeerd in het systeem. Om de redundantie van de Presburger formules te beslissen, werd Omega+ gebruikt, een herwerking van de eerder ontwikkelde Omega-library. De implementatie van het algoritme is volledig functioneel, en heeft al bewezen in sommige gevallen (zie bijvoorbeeld het probleem van de query op het einde van hoofdstuk 3) het systeem toe te staan nieuwe problemen op te lossen. Bij het testen van ons algoritme, waren de resultaten bij veel testen echter niet zoals we gehoopt hadden.

Er zijn verschillende problemen, zoals “Bin Packing”, waar ons algoritme duidelijk een verbetering teweeg brengt, maar anderzijds zijn er ook problemen waar het niets doet, en in sommige gevallen zelfs het IDP-systeem vertraagt. Een mogelijke reden hiervoor is dat het algoritme veel zou kunnen verbeteren tijdens het gebruik van grenzen, maar dat dit niet ten volle benut wordt. We hebben enkele experimenten gedaan, waar werkelijk uit bleek dat ons algoritme bij het gebruik van grotere FOBDDs er wel in slaagt de efficiëntie

te verhogen.

## 5.4 Verder onderzoek

Zoals hierboven al is aangehaald zijn er nog verschillende aspecten te verbeteren aan de implementatie van het algoritme en het gebruik ervan, om de efficiëntie ervan te verhogen. Een eerste mogelijkheid is een grondig onderzoek naar andere aspecten in het systeem waar het versimpelen van FOBDDs een grote invloed kan hebben. Een mogelijk aspect wat van naderbij bekeken kan worden is het vereenvoudigen van FOBDDs die gebruikt worden voor het herkennen van equivalente subformules (hoofdstuk 3.2.2). Hier is de kans groot dat ons algoritme van nut kan zijn, maar herkennen van die formules is zelf nog in ontwikkeling, en niet volledig operatief in het systeem.

Enkele andere mogelijke verbeteringen liggen bij het algoritme zelf. Om dit efficiënter te laten lopen, kan het misschien een goed idee zijn om een heuristiek te ontwikkelen, die uitzoekt hoeveel aritmetiek een bepaalde FOBDD bevat, en wat de kans is dat we aritmetische redundantie in die FOBDD gaan vinden. Een heel eenvoudige heuristiek zou al kunnen zijn, dat we uitzoeken of een FOBDD überhaupt aritmetiek bevat. Zo vermijden we ons algoritme te laten lopen op grote FOBDDs, waarvan we op voorhand al weten dat we ze toch niet gaan kunnen vereenvoudigen, omdat er geen aritmetiek instaat. Een ander aspect dat we kunnen onderzoeken is de beslisser die we gebruikt hebben. In onze implementatie werd oorspronkelijk Omega gebruikt om de beslissingen te maken, en dit werd later vervangen door Omega+, welke een nieuwere herwerkte versie is van Omega. Een mogelijkheid tot verder onderzoek zou kunnen zijn om te onderzoeken hoe andere beslissers voor Presburger aritmetiek presteren, en eventueel Omega+ hierdoor te vervangen. Een systeem dat in de context van deze thesis maar kort bekeken is, en dat misschien potentieel zou kunnen hebben, is het ISL systeem ([Ver10]).

In het IDP-systeem wordt gewerkt met een uitbreiding van eerste-orde logica, namelijk  $FO(.)$ . Er is al een simpele uitbreiding hiervoor geformuleerd en geïmplementeerd, maar  $FO(.)$  bevat nog meer concepten waar we de bruikbaarheid van ons algoritme naar zouden kunnen uitbreiden. Zo bevat  $FO(.)$  bijvoorbeeld aggregaten, functies op verzamelingen, die in FOBDDs kunnen worden voorgesteld. Als dit functies zijn op verzamelingen van gehele getallen, of functies zoals kardinaliteit, kunnen we hier misschien informatie uithalen om nog meer redundantie uit een FOBDD te elimineren.

# Bibliografie

- [And97] H.R. Andersen. An introduction to binary decision diagrams. *Lecture notes, available online, IT University of Copenhagen*, (October 1997):30, 1997.
- [ASU86] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. Compilers: principles, techniques, and tools. August 1986.
- [BBJ02] George S Boolos, John P Burgess, and Richard C Jeffrey. Computability and Logic (fourth edition). *American Mathematical Monthly*, 2002.
- [Bry86] R.E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.
- [BW03] David Basin and Burkhart Wolff, editors. *Theorem Proving in Higher Order Logics*, volume 2758 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.
- [Che09] Chun Chen. Omega+ library, 2009.
- [CL01] René Cori and Daniel Lascar. *Mathematical Logic: A Course with Exercises*. Oxford University Press, 2001.
- [Coo71] Stephen A. Cook. The Complexity of Theorem-Proving Procedures. In *Proceedings of the third annual ACM symposium on Theory of computing - STOC '71*, pages 151–158, New York, New York, USA, May 1971. ACM Press.
- [dB72] N.G de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indagationes Mathematicae (Proceedings)*, 75(5):381–392, January 1972.
- [Die05] Reinhard Diestel. *Graph Theory, Graduate Texts in Mathematics, Volume 173*. Springer-Verslag, Heidelberg, 2005.
- [DT04] Marc Denecker and Eugenia Ternovska. Inductive Situation Calculus. In Didier Dubois, Christopher A Welty, and Mary-Anne Williams, editors, *KR*, pages 545–553. AAAI Press, 2004.
- [Een04] Niklas Een, Niklas and Sorensson. An extensible SAT-solver. *Theory and Applications of Satisfiability Testing*, pages 502–518, 2004.
- [End72] Herbert B Enderton. *A Mathematical Introduction To Logic*. Academic Press, 1972.

- [GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994.
- [God31] Kurt Godel. ber formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik*, 38(1):173–198, 1931.
- [Gou95] Jean Goubault. A BDD-Based Simplification and Skolemization Procedure. *Logic Journal of IGPL*, 3(6):827–855, 1995.
- [KUL12] KULeuven Knowledge Representation and Reasoning research group. The idp system reference manual. pages 1–15, 2012.
- [Lib04] Leonid Libkin. *Elements Of Finite Model Theory*. Springer, 2004.
- [MVFH03] Yoram Moses, Moshe Y Vardi, Ronald Fagin, and Joseph Y Halpern. *Reasoning About Knowledge*. MIT Press, 2003.
- [MWD06] Maarten Mariën, Johan Wittocx, and Marc Denecker. The IDP framework for declarative problem solving. In *Search and Logic Answer Set Programming and SAT*, pages 19–34, 2006.
- [Pre29] Mojzesz Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In F Leja, editor, *welchem die Addition als einzige Operation hervortritt Compte Rendus du I Congrks des Mathematiciens des pays Slavs Warsaw*, pages 92–101. Warsaw, 1929.
- [Rud93] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Proceedings of the 1993 IEEE/ACM international conference on Computer-aided design*, pages 42–47. IEEE Computer Society Press, 1993.
- [Sto74] Larry Joseph Stockmeyer. The complexity of decision problems in automata theory and logic., 1974.
- [Tse68] G S Tseitin. On the Complexity of Derivation in Propositional Calculus. *Studies in Constructive Mathematics and Mathematical Logic {II}*, 8:234–259, 1968.
- [Ver10] Sven Verdoolaege. ISL: An integer set library for the polyhedral model. In Komei Fukuda, Joris van der Hoeven, Michael Joswig, and Nobuki Takayama, editors, *Lecture Notes in Computer Science*, pages 299–302. Springer, 2010.
- [Wit10] Johan Wittocx. *Finite Domain and Symbolic Inference Methods for Extensions of First-Order Logic*. PhD thesis, Katholieke Universiteit Leuven, Leuven, Belgium, January 2010.
- [WMD10] Johan Wittocx, Maarten Mariën, and Marc Denecker. Grounding FO and FO (ID) with Bounds. *Journal of artificial intelligence research*, 38:223–269, 2010.



# Bijlage A

## Getypeerde eerste-orde logica

**Definitie A.1.** Een formule in eerste-orde logica over een vocabularium  $\Sigma$ , is van de vorm:

$$\phi, \psi := \neg\phi \quad (\text{negatie}) \quad (\text{A.1})$$

$$(\phi \wedge \psi) \quad (\text{conjunctie}) \quad (\text{A.2})$$

$$(\phi \vee \psi) \quad (\text{disjunctie}) \quad (\text{A.3})$$

$$\forall x(\phi) \quad (\text{Universele kwantificatie}) \quad (\text{A.4})$$

$$\exists x(\phi) \quad (\text{Universele kwantificatie}) \quad (\text{A.5})$$

$$P(t_1, t_2, \dots, t_n) \quad (\text{Predicaat Toepassen}) \quad (\text{A.6})$$

$$t_1 = t_2 \quad (\text{Gelijkheidsrelatie}) \quad (\text{A.7})$$

Met  $x \in \Sigma_{var}$ ,  $\text{type}(t_i) = s_i$ ,  $P(s_1, \dots, s_n) \in \Sigma_{pred}$  en alle  $t_i$  termen, welke gedefinieerd worden door:

$$t := x \quad (\text{een variabele is een term}) \quad (\text{A.8})$$

$$F(t_1, t_2, \dots, t_n) \quad (\text{Functie toepassen}) \quad (\text{A.9})$$

Met alle  $t_i$  termen,  $\text{type}(t_i) = s_i^1$  en  $F(s_1, \dots, s_n) : s \in \Sigma_{func}$ . We noemen verder een formule van type (1.6) of (1.7) ook wel een **atomische formule**. De verzameling van alle formules in de eerste-orde logica over een vocabularium  $\Sigma$  noteren we met  $\mathcal{L}_{\mathbf{FO}, \Sigma}$

**Definitie A.2.** Een  $\Sigma$ -structuur  $I$  is een afbeelding die:

- elke  $s \in \Sigma_{type}$  op een niet-lege verzameling  $D_s$  afbeeldt. We noemen  $D_s$  het domein van  $s$ , ofwel de interpretatie van  $s$ , en noteren het ook wel als  $s_I$
- elke  $P(s_1, \dots, s_n) \in \Sigma_{pred}$  op een relatie  $R_P \subset (D_{s_1} \times D_{s_2} \times \dots \times D_{s_n})$  afbeeldt. We noemen  $R_P$  de interpretatie van  $P$ , en noteren het ook als  $P^I$ .
- elke  $F(s_1, \dots, s_n) : s \in \Sigma_{func}$  op een functie  $G_F \subset (D_{s_1} \times D_{s_2} \times \dots \times D_{s_n}) \rightarrow D_s$  afbeeldt. We noemen  $G_F$  de interpretatie van  $F$ , en noteren het ook als  $F^I$ .

---

<sup>1</sup>We breiden de type-afbeelding uit voor termen, door te zeggen dat  $\text{type}(F(t_1, t_2, \dots, t_n))$  bepaald wordt door het type van het resultaat van de functie.

We noemen een  $\Sigma$ -structuur  $I$  eindig als  $D_s$  eindig is voor alle  $s \in \Sigma_{type}$ .

**Definitie A.3.** Een variabele-interpretatie  $\omega$  voor  $I$  ( $\Sigma$ -structuur) is een afbeelding die elke  $x \in \Sigma_{var}$  met  $type(x) = s$  op een  $d \in D_s$  afbeeldt. Aangezien het voor elk element in het vocabularium duidelijk is, of we het moeten interpreteren met  $I$ , of met  $\omega$  noteren we  $I\omega$  voor de afbeelding vanuit  $\Sigma$  die elk element interpreteert. We noemen deze afbeelding dan ook kortweg de interpretatie.

Semantiek voor termen:

$$\begin{aligned} I\omega(x) &= \omega(x) \\ I\omega(F(t_1, \dots, t_n)) &= G_F(I\omega(t_1), \dots, I\omega(t_n)) \end{aligned}$$

Semantiek voor formules:

$$\begin{aligned} I\omega \models P(t_1, \dots, t_n) & \text{ als } (I\omega(t_1), I\omega(t_2), \dots, I\omega(t_n)) \in R_P \\ \models t_1 = t_2 & \text{ als } I\omega(t_1) = I\omega(t_2) \\ \models \neg\phi & \text{ als } I\omega \not\models \phi \\ \models \phi_1 \wedge \phi_2 & \text{ als } I\omega \models \phi_1 \text{ en } I\omega \models \phi_2 \\ \models \phi_1 \vee \phi_2 & \text{ als } I\omega \models \phi_1 \text{ of } I\omega \models \phi_2 \\ \models \forall x(\phi) & \text{ als } I\omega[x/d] \models \phi \text{ voor alle } d \in D_s, \text{ met } s \text{ het type van } x \\ \models \exists x(\phi) & \text{ als er een } d \in D_s \text{ bestaat zodat } I\omega[x/d] \models \phi, \text{ met } s \text{ het type van } x \end{aligned}$$

## Bijlage B

### Populariserende samenvatting

## Het vereenvoudigen van FOBDDs door middel van Presburger aritmetiek

Het doel van deze thesis is het verbeteren van de efficiëntie van een modelexpansie algoritme. Dit is een algoritme voor het oplossen van problemen zoals een sudoku. Een sudoku kunnen we formeel specificeren als volgt: We hebben een vocabularium  $\Sigma$  (bv. vakjes, nummers, rijen, kolommen, ...), en een eindige verzameling regels  $\mathcal{T}$  waar de elementen uit dat vocabularium aan moeten voldoen (bv. “Als twee vakjes tot dezelfde rij behoren, bevatten ze niet hetzelfde nummer.”) Hierbij hebben we voor een deelvocabularium  $\sigma$  van dat vocabularium een gegeven interpretatie (bv. “Op vakje 3 is er een 9 ingevuld.”). De puzzel bestaat er dan in de interpretatie van  $\sigma$  uit te breiden tot een interpretatie voor  $\Sigma$  die voldoet aan  $\mathcal{T}$ . Gelijkaardige problemen, zijn puzzels zoals de battleship-puzzel, of meer reële probleemstellingen zoals het plannen van examenroosters of het opstellen van een treinschema. We noemen dit soort problemen modelexpansieproblemen.

Bij het oplossen van dit soort problemen, moet de computer zelf zijn oplossingsstrategie bepalen. De meest voor de hand liggende strategie is natuurlijk elke mogelijke oplossing genereren, en dan kijken of die voldoen aan de voorwaarden. Dit is uiteraard niet optimaal. Er worden dan namelijk enorm veel oplossingen getest, waarvan we op voorhand al weten dat ze niet voldoen. Het is dus belangrijk dat de computer een zo efficiënt mogelijke oplossingsstrategie voor het probleem kan bepalen. Het bedenken en verbeteren van systemen die voor grote verzamelingen problemen efficiënte strategieën kunnen bepalen vormt een groot onderzoeksgebied. De werkgroep KRR van de afdeling DTAI van het departement computerwetenschappen aan de KU Leuven is hier onder meer mee bezig. Er is een systeem genaamd IDP ontwikkeld dat modelexpansieproblemen zo efficiënt mogelijk oplost.

Als we tijdens het uitvoeren van een modelexpansie meerdere malen dezelfde (deel-)formule tegenkomen, willen we van deze informatie gebruik kunnen maken, en eerder berekende resultaten kunnen hergebruiken. Om dit te kunnen doen, moeten we eerst en vooral ontdekken dat twee formules equivalent zijn. Het IDP systeem gebruikt hiervoor speciale normaalvorm van formules, die FOBDDs (Binary Decision Diagrams) worden genoemd. Hoewel FOBDDs hier erg goed in zijn, blijkt toch dat ze niet perfect zijn, en niet altijd kunnen detecteren of twee formules equivalent zijn. Eén van de problemen bevindt zich in het gebruik van arithmetiek in FOBDDs. Er geldt dat de beweringen “Voor alle getallen groter dan 3 geldt P” en “Voor alle getallen groter dan 3 en groter dan 2 geldt P”, equivalent zijn, maar dat kunnen we niet kunnen ontdekken met FOBDDs.

Voor deze thesis werd een algoritme ontwikkeld om dit probleem uit de wereld te helpen. Het bleek echter theoretisch niet mogelijk te zijn om dit met alle aritmetische formules te doen. We hebben ons beperkt tot formules die geen vermenigvuldiging bevatten. We gebruiken en bewijzen het theoretisch resultaat dat voor twee aritmetische formules  $\varphi$  en  $\psi$  die geen vermenigvuldiging bevatten, we altijd in eindige tijd kunnen beslissen of ze equivalent zijn. Deze informatie werd gebruikt om de gebruikte FOBDDs in het modelexpansie algoritme verder te verbeteren, zodat de aritmetische redundantie in deze FOBDDs tot een minimum beperkt wordt.