KU LEUVEN

FACULTEIT INGENIEURSWETENSCHAPPEN

Orakels in IDP – Een implementatie voor epistemische logica

Neline van Ginkel

Thesis voorgedragen tot het behalen van de graad van Master of Science in de ingenieurswetenschappen: computerwetenschappen

Promotor:
Prof. dr. M. Denecker

Assessoren:
Bart Bogaerts
Prof. dr. ir. F. Piessens

Begeleider:
Bart Bogaerts

Academiejaar 2014 – 2015

Voorwoord

Het afgelopen jaar heb ik veel tijd en moeite gestoken in deze thesis en er enorm veel uit bijgeleerd. Deze thesis zou echter niet mogelijk geweest zijn zonder alle mensen rond mij. In dit voorwoord zou ik daarom graag iedereen willen bedanken die er voor mij was het afgelopen jaar tijdens alle aspecten van mijn thesis en daarbuiten.

Ik wil Jan bedanken, omdat hij er altijd voor mij was. Bedankt voor alle momenten van ontspanning en het begrip als ik langer moest doorwerken. Ik wil ook mijn ouders en schoonouders bedanken voor alle begrip en hulp.

Ik wil iedereen van de onderzoeksgroep KRR bedanken, meer in het specifiek professor Denecker en Bart Bogaerts, om mij gedurende het jaar te begeleiden, mij verder te helpen en te voorzien van feedback.

Ik wil ook mijn vrienden bedanken. Steven: bedankt voor alle hulp, tips, en toffe gesprekken. Tim en Bart: bedankt voor alle toffe momenten!

Het afgelopen jaar heb ik ook ontspanning gevonden in het meedoen aan capture-the-flag wedstrijden. Aan alle Hacknam Style members: bedankt voor alle toffe CTF's!

Iedereen die in meer of mindere mate mij heeft ondersteund het afgelopen jaar: bedankt!

Neline van Ginkel

Inhoudsopgave

Samenvatting

Epistemische logica is een krachtige logica om te redeneren met het bezitten of niet bezitten van kennis. In deze thesis onderzoeken we hoe we een subset van deze logica, geordende epistemische logica kunnen toevoegen aan het IDP kennisbank systeem, een systeem waarin we kennis kunnen representeren en diverse inferentietaken kunnen uitvoeren op deze kennis.

We ondersteunen deze geordende epistemische logica door gekwantificeerde booleaanse formules (QBF) toe te voegen aan de SAT-solver MinisatID. We stellen geneste ECNF voor, een uitbreiding van de invoertaal ECNF. We introduceren ook een algoritme dat deze geneste ECNF kan oplossen en bespreken hier verschillende varianten van.

Het resultaat is een krachtige solver die gekwantificeerde booleaanse formules combineert met bestaande expressieve concepten zoals inductieve definities, aggregaten en aritmetica.

Lijst van afkortingen

Afkortingen

CNF	Conjunctieve Normaalvorm / Conjunctive Normal Form
CDCL	Conflict-Driven Clause Learning
ECNF	Extended Conjunctieve Normaalvorm / Extended Conjunctive Normal Form
SAT	Satisfiability
QBF	Gekwantificeerde Booleaanse Formule / Quantified Boolean Formulas

Hoofdstuk 1

Inleiding

Kennisrepresentatie is het domein dat zich bezig houdt met de voorstelling van kennis en het kunnen redeneren hiermee. IDP is een kennisbanksysteem, een systeem waarin we kunnen redeneren met kennis. [Mariën et al., 2006] IDP heeft als invoertaal "de IDP taal", een expressieve declaratieve invoertaal. Het IDP kennisbanksysteem bestaat uit twee delen, de *grounder* en de *solver*[De Cat, 2014]. De grounder vertaalt een probleem in de IDP taal naar een taal op een lager niveau: ECNF. Deze ECNF bevat een verzameling van logische constraints over een verzameling van variabelen. De solver leest ECNF in en zoekt een oplossing voor het gegeven probleem.

Epistemische logica [Davis and Morgenstern, 1983] is een logica waarin we kunnen redeneren met kennis. Het IDP kennisbanksysteem heeft nog geen ondersteuning voor deze logica. In deze thesis onderzoeken we hoe we epistemische logica kunnen toevoegen aan het IDP kennisbanksysteem.

Het toevoegen van epistemische logica aan het IDP kennisbanksysteem bestaat in de gekozen oplossing uit meerdere delen: de IDP taal moet uitgebreid worden, de grounder die de IDP taal vertaalt naar ECNF moet deze uitgebreide IDP taal kunnen vertalen en de solver moet uitgebreid worden. In deze thesis beperken we ons tot het ondersteunen van epistemische logica in de solver, MinisatID. We bekijken dit vooral vanuit een toegepast standpunt en gaan minder in op het wiskundig formele aspect hiervan.

We kunnen geordende epistemische logica [Vlaeminck et al., 2012a], een subset van epistemische logica, op een eenvoudige manier vertalen naar gekwantificeerde booleaanse formules (QBF). We voegen daarom QBF toe aan ECNF, de invoertaal van de solver MinisatID. Het resultaat is een krachtige solver die gekwantificeerde booleaanse formules combineert met expressieve taalconcepten zoals inductieve definities, aggregaten en aritmetica. Door QBF toe te voegen aan ECNF ondersteunen we niet enkel geordende epistemische logica, maar kunnen we ook andere logica's zoals hogere-orde logica combineren met de krachtige concepten uit het IDP kennisbanksysteem.

1.1 Overzicht

In hoofdstuk 2 leggen we uit wat het kennisbanksysteem IDP is en hoe de SAT-solver onderliggend hieraan werkt. In hoofdstuk 3 bekijken we wat epistemische logica precies is en geven we diverse voorbeelden die aangeven dat het uitbreiden van eerste-orde logica met een epistemische operator de natuurlijke expressiviteit verhoogt. Hierdoor kunnen we problemen op meer natuurlijke wijze verwoorden. Aangezien het IDP kennisbanksysteem niet met klassieke eerste-orde logica werkt, maar met de "de IDP taal", een meer expressieve invoertaal, zal de uitbreiding met een epistemische operator de mogelijkheid geven om meer problemen te modelleren en op te lossen dan bestaande systemen die epistemische logica ondersteunen.

In hoofdstuk 4 leggen we uit wat gekwantificeerde booleaanse formules (QBF) zijn, wat geldige oplossingen zijn voor dergelijke formules en hoe bestaande QBF-solvers aan deze oplossingen komen. We kijken hier vooral naar QBF-solvers die werken met behulp van Conflict-Driven Clause Learning technieken.

In hoofdstuk 5 beschrijven we in meer detail de probleemstelling van deze thesis. In hoofdstuk 6 beschrijven we hoe we geordende epistemische logica kunnen voorstellen in ECNF, de invoertaal van MinisatID. Een mogelijke manier om epistemische logica op te lossen is door deze logica te vertalen naar QBF (gekwantificeerde booleaanse formules). Bestaande QBF-solvers kunnen echter geen inductieve definities, aggregaten, aritmetica en andere uitbreidingen aan. MinisatID ondersteunt deze uitbreidingen wel, maar ondersteunt nog geen QBF. We voegen QBF toe in de vorm van geneste ECNF. In hoofdstuk 7 bekijken we hoe we geneste ECNF kunnen oplossen in MinisatID. We stellen een algoritme voor dat geneste ECNF op een hiërarchische manier oplost. Dit algoritme staat orthogonaal op bestaande taalconcepten in MinisatID, zoals inductieve definities, aggregaten en aritmetica.

In hoofdstuk 8 trekken we conclusies en bespreken we welk toekomstig werk er nog uit deze thesis kan volgen.

Hoofdstuk 2

IDP - een kennisbanksysteem

In dit hoofdstuk geven we een uitleg rond kennisbanksystemen, waar we vooral focussen op het IDP kennisbanksysteem. Eerst geven we een algemene uitleg rond kennisbanksystemen waarin we bespreken wat een kennisbanksysteem is en wat het doel is van een dergelijk systeem. Daarna gaan we in op het IDP kennisbanksysteem en bekijken we hoe dit systeem in elkaar zit. Ook kijken we naar een ander kennisbanksysteem, Clingo. Vervolgens bekijken we wat een SAT-solver is, een belangrijke component in het IDP kennisbanksysteem. We leggen uit hoe SAT-solvers in het algemeen te werk gaan en beschrijven de in- en uitvoer van de SAT-solver MinisatID, de SAT-solver die we in het vervolg van deze thesis gaan uitbreiden.

2.1 Kennisbanksystemen

Een kennisbanksysteem is een systeem dat kennis gebruikt om hierop verschillende generieke inferentiemethodes uit te voeren. [De Cat, 2014] Kennisbanksystemen bestaan uit twee componenten: een taal om op een gemakkelijke en natuurlijke manier kennis te representeren en een aantal inferentiemethodes: verschillende methodes die allemaal dezelfde kennis kunnen gebruiken om computationele problemen op te lossen.

De invoer voor kennisbanksystemen bestaat uit verschillende componenten. Een vocabularium bevat de symbolen waar een theorie of structuur over kan spreken, namelijk types, predicaten en functies. Een logische theorie is altijd gedefinieerd over een vocabularium. In een dergelijke theorie kunnen we de algemene kennis van een domein beschrijven. Een structuur is een toekenning van waarden aan verschillende predicaten en functies uit een vocabularium. We kunnen twee theorieën, twee structuren of twee vocabularia samenvoegen tot één. Zo kunnen we kennis op een logische manier opdelen en groeperen. Een term neemt als invoer een structuur over een vocabularium en kent hier een waarde aan toe.

Een voorbeeld van kennis die we in een kennisbank kunnen opslaan is informatie rond uurroosters voor studenten. In de kennisbank kunnen we algemene kennis opslaan, zoals “Een professor mag geen twee lessen geven op hetzelfde moment”, maar ook meer specifieke kennis als “De les ‘Geschiedenis van de Informatica’ is

op maandag". Met behulp van de kennis in die kennisbank kunnen we diverse vraagstukken oplossen, zoals "Heeft deze student op maandag les?", "Voldoet het uurrooster dat ik heb opgesteld wel aan de regels die gelden voor een uurrooster?" en "Het uurrooster dat ik heb opgesteld voldoet niet aan de regels. Welke veranderingen moet ik aan het uurrooster doorvoeren om te zorgen dat het wel aan deze regels voldoet?". Al deze vraagstukken kunnen we met behulp van hetzelfde systeem en dezelfde kennis oplossen.

Een ander voorbeeld van kennis die we in een kennisbank kunnen opslaan is informatie voor een robot die moet navigeren in een bepaalde ruimte. In de kennisbank kunnen we algemene informatie opslaan, zoals "De robot mag niet in het ravijn rijden", maar ook kennis zoals "De robot bevindt zich op locatie (1,0)". Hiermee kan de kennisbank vragen beantwoorden zoals "Op welke locatie bevindt de robot zich als hij zich één plaats naar voren beweegt", "Welke stappen moet de robot doen om locatie (3,3) te bereiken?" en "De huidige route zou de robot door de onbereikbare locatie (2,2) laten gaan, wat is de minimale aanpassing aan de route om de robot zijn bestemming alsnog te laten bereiken?"

2.1.1 Inferentiemethodes

Inferentiemethodes zijn methodes waarmee we verschillende computationele problemen kunnen oplossen met behulp van dezelfde kennis. We beschrijven een aantal inferentiemethodes en geven voorbeelden in welke situatie we een inferentiemethode kunnen gebruiken.

Modelexpansie is een inferentiemethode waarbij we een logische theorie en een partiële structuur over een bepaald vocabularium als invoer nemen en een structuur over het volledige vocabularium als uitvoer geven die een uitbreiding is van de partiële structuur en voldoet aan de theorie. [Mariën et al., 2006] Deze uitgebreide structuur noemen we ook wel een model. Als we een structuur hebben die aangeeft welke vakken een student volgt en wanneer de lessen van alle vakken zijn, en we ook een theorie hebben die de algemene regels rond uurroosters bevat, kunnen we met modelexpansie het lesrooster van de student genereren. Ook kunnen we diverse problemen voor de robot oplossen met modelexpansie. Als we een structuur hebben waarin de plattegrond voor de robot bekend is en zijn start- en doelpositie, en we hebben een theorie waarin beschreven staat welke acties de robot kan nemen, kunnen we hiermee een sequentie van stappen genereren zodat de robot zijn doelpositie bereikt. Als we een onvolledige plattegrond hebben, kunnen we met modelexpansie alle mogelijke plattegronden genereren, door alle modellen op te vragen.

Voor andere problemen kunnen we gebruik maken van *satisfiability checking*. Hierin berekenen we of er een model bestaat dat voldoet aan de theorie en de structuur, dus of de theorie *satisfiable* is met de gegeven structuur. Ook hier moet de structuur niet volledig zijn, maar kunnen we ook rekenen met een partiële structuur. In tegenstelling tot modelexpansie moet het kennisbanksysteem hier geen model teruggeven, maar moet dit enkel berekenen of er een model bestaat. Als we een structuur hebben waarin we een specifiek uurrooster beschrijven en we hebben een theorie waarin we beschrijven aan welke regels een geldig uurrooster voldoet, kunnen

we met satisfiability checking bepalen of het gegeven uurrooster een geldig uurrooster is.

Modelrevisie is een derde inferentiemethode. Hierin hebben we een volledige structuur en een theorie als invoer, waarbij de structuur niet voldoet aan de theorie. We zoeken een minimale aanpassing aan de structuur zodat deze wel voldoet aan de theorie. Als we een volledig uurrooster hebben, maar we hebben extra restricties zodat het uurrooster niet meer voldoet aan de theorie, kunnen we met modelrevisie een uurrooster genereren dat zo dicht mogelijk ligt bij het originele uurrooster, maar wel voldoet aan de extra voorwaarden.

Propagatie is een inferentiemethode waarin we alle gevolgen van een theorie op een partiële structuur kunnen vinden. Het kennisbanksysteem maakt hierin geen keuzes, zoals bij modelexpansie, maar zal de structuur enkel uitbreiden met alle directe gevolgen van de theorie.

Minimalisatie is een specialisatie van modelexpansie, waarin we naast een theorie en een structuur ook een term als invoer geven. We zoeken een structuur zodat de term minimaal is en er dus geen andere structuur is die voldoet aan de theorie, waarvoor de waarde van de term kleiner is. Een voorbeeld is de eerder genoemde robot die zijn weg moet zoeken met behulp van een plattegrond. Met minimalisatie kunnen we op zoek gaan naar de kortste weg naar het doel, in plaats van enkel op zoek te gaan naar een willekeurige weg naar het doel.

Progressie is een inferentiemethode bedoeld voor logische theorieën in lineaire tijd calculus, een specifieke taal waarin we eenvoudig dynamische systemen kunnen modelleren. We kunnen hiermee de robot simuleren, door eerst zijn beginsituatie te geven en daarna telkens één stap in de simulatie te zetten. [Bogaerts et al., 2014]

In al deze inferentiemethodes kunnen we gebruik maken van dezelfde logische theorieën, ondanks het feit dat de taak die we willen uitvoeren in ieder geval verschillend is. Op die manier maken we een strikte scheiding tussen de kennis (in een logische theorie) en de specifieke taak die we met de kennis willen uitvoeren, die onder andere afhangt van een (partiële) structuur. Door deze scheiding kunnen nieuwe taken toegevoegd aan het kennisbanksysteem, zonder dat we de kennis opnieuw moeten schrijven. Als de kennis zelf verandert, moeten we de theorie of de structuur aanpassen, maar kunnen de inferentiemethodes onaangepast blijven.

2.2 Het IDP kennisbanksysteem

IDP is een kennisbanksysteem dat “de IDP taal” als invoertaal gebruikt, een taal waarin we eenvoudig kennis in kunnen representeren. De IDP taal is een uitbreiding van eerste-orde logica met onder andere inductieve definities, types, aggregaten en aritmetica. Het IDP kennisbanksysteem zullen we verder in deze thesis gebruiken en uitbreiden. IDP lost de meeste inferentiemethodes op met behulp van een grounder en een solver. De grounder zal de invoer in de IDP taal eerst vertalen naar ECNF, de invoertaal van de solver. Deze taal is eenvoudiger op te lossen voor de solver. De solver MinisatID neemt ECNF als invoer en probeert hiervoor een model te genereren. ECNF bevat onder andere constraints voor inductieve definities, aggregaten en

aritmetica met gehele getallen. We bekijken SAT-solvers in meer detail in sectie 2.4. [De Cat, 2014]

IDP ondersteunt verschillende inferentiemethodes, waarvan de voornaamste modelexpansie, propagatie, satisfiability checking, minimalisatie en progressie zijn. IDP combineert de declaratieve taal "de IDP taal" en de imperatieve taal Lua. Hierdoor kunnen we logische theorieën schrijven in de IDP taal, maar kunnen we procedures om verschillende inferentiemethodes op te roepen en te combineren schrijven in Lua. [Pooter et al., 2011]

2.3 Clingo

Clingo is een kennisbanksysteem met Answer Set Programming (ASP) als invoertaal. ASP is nauw verwant aan de IDP taal. Clingo ondersteunt modelexpansie, progressie en minimalisatie. We kunnen de werking van Clingo controleren met behulp van de programmeertalen Python en Lua. Deze interface is vooral gericht op controle over het grounden en solven van ASP programma's. Clingo heeft evenals IDP een grounder en een solver, maar deze zijn samengevoegd tot één proces, waardoor er geen constante afwisseling is tussen het oproepen van de grounder en de solver. [Gebser et al., 2014]

2.4 SAT-solvers

Een SAT-solver lost satisfiability-problemen (SAT-problemen) voor propositionele logica op. In een satisfiability-probleem zoeken we of er een toekenning van waarden aan variabelen voor een formule in propositionele logica bestaat. Een voorbeeld van een dergelijke formule is:

$$(\text{links} \vee \text{rechts}) \wedge (\neg\text{links} \vee \neg\text{rechts}) \tag{2.1}$$

Definitie 1: Propositionele logica is gedefinieerd als volgt:

- Een propositioneel atoom p is een zin.
- Als ϕ een zin is, dan is $\neg\phi$ (niet ϕ) ook een zin.
- Als ϕ en ψ zinnen zijn, dan zijn de volgende expressies ook zinnen:
  - $\phi \wedge \psi$: ϕ en ψ
  - $\phi \vee \psi$: ϕ of ψ
  - $\phi \implies \psi$: Als ϕ waar is, moet ψ ook waar zijn
  - $\phi \iff \psi$: Ofwel zijn ϕ en ψ beide waar, ofwel zijn beide formules onwaar

Een literal is een atoom of zijn negatie, zoals links of ¬rechts. We definiëren een vocabularium Σ als de verzameling van alle propositionele atomen p. Deze

propositionele atomen noemen we ook wel variabelen. Een disjunctie van één of meerdere literals is een clause. Een formule in de conjunctieve normaalvorm (CNF) bestaat uit een conjunctie van clauses. Een geldige toekenning van waarden aan de bovenstaande formule is links $\wedge \neg$rechts. De bovenstaande formule is dus satisfiable.

We kunnen alle formules in propositionele logica omzetten naar de conjunctieve normaalvorm. We passen hiervoor de volgende regels toe:

- We elimineren equivalenties: we herschrijven $\phi \iff \psi$ naar $(\phi \implies \psi) \wedge (\psi \implies \phi)$
- We elimineren implicaties: we herschrijven $\phi \implies \psi$ naar $\neg\phi \vee \psi$
- We brengen negaties zoveel mogelijk naar binnen:
  - We herschrijven $\neg(\phi \wedge \psi)$ naar $(\neg\phi \vee \neg\psi)$ met de wetten van de Morgan.
  - We herschrijven $\neg(\phi \vee \psi)$ naar $\neg\phi \wedge \neg\psi$ met de wetten van de Morgan.
- We elimineren dubbele negaties: we herschrijven $\neg\neg\phi$ naar ϕ
- We brengen disjuncties naar binnen: we herschrijven $(\phi \wedge \psi) \vee \delta$ naar $(\phi \vee \delta) \wedge (\psi \vee \delta)$

De omzetting van een willekeurige formule naar de conjunctieve normaalvorm zorgt ervoor dat de resulterende formule veel groter wordt: het herschrijven van equivalenties en het naar binnen brengen van disjuncties verdubbelt telkens de grootte van de relevante formule.

Een SAT-solver krijgt als invoer een formule in de conjunctieve normaalvorm en geeft als uitvoer een toekenning van waarden aan iedere variabele zodat de formule voldaan is, indien er een geldige toekenning bestaat. In het andere geval is de uitvoer 'unsatisfiable'.

Het Davis-Putnam-Logemann-Loveland algoritme (DPLL) [Davis et al., 1962] is een compleet, backtrack-gebaseerd algoritme om de satisfiability van een formule in CNF te bepalen. Het idee van het DPLL algoritme is om de zoekboom op een gestructureerde manier te doorzoeken met behulp van chronologische backtracking. Om de volledige CNF-formule waar te maken, moet er voor iedere clause van die formule minstens één literal waar zijn. Als er nog geen enkele literal van een clause *true* is, en er nog maar één literal onbekend is, kunnen we propageren dat de laatste literal van deze clause *true* moet zijn. Het propageren met nog maar één onbekende literal in een clause heet unit propagatie. Als er geen unit propagaties meer mogelijk zijn, zal het DPLL-algoritme een willekeurig atoom dat nog geen waarde heeft kiezen en hier een waarde aan toekennen. Als er een conflict is, zal DPLL backtracken over eerder gemaakte keuzes. Een conflict houdt in dat er een clause bestaat waarvoor alle literals $false$ zijn. Deze clause noemen we een conflict clause.

Conflict-Driven Clause Learning (CDCL) [Silva et al., 2009] is een verbetering op dit DPLL-algoritme. CDCL zal delen van de zoekboom overslaan waar zeker geen oplossing te vinden is. Iedere literal krijgt een waarde door unit propagatie of door een keuze. Als we voor iedere literal die gepropageerd is bijhouden door welke

clause zijn propagatie gebeurde, kunnen we hiermee bepalen door welke keuzes deze propagatie uiteindelijk is veroorzaakt. CDCL zal het conflict analyseren en probeert uit dit conflict te leren. CDCL doet dit door het conflict expliciet te maken met behulp van een explanation clause. Deze clause legt uit wat de reden van het conflict is. De SAT-solver genereert deze clause op basis van de conflict clause en de clauses van de eerder gepropageerde literals tot het laatste relevante keuzepunt. De generatie van deze clause gebeurt met behulp van resolutie, dat twee clauses samenvoegt tot één nieuwe clause. Een explanation clause is altijd een logisch gevolg van de originele CNF formule. De pseudocode van het Conflict-Driven Clause Learning algoritme is terug te vinden in Algoritme 1.

Algorithm 1 Conflict-Driven Clause Learning algoritme

```
function ConflictDrivenClauseLearning(Theory T = {c_1 ... c_n})
    decisionlevel ← 0
    for all variabele : variabelen do
        variabele ← unknown
    end for
    while not all variables assigned do
        conflict ← unitpropagatie()
        if conflict then
            level, explanation ← analyzeConflict(Theory T, variabelen)
            if level == 0 then
                return UNSATISFIABLE
            else
                backtrack(level)
                addClause(explanation)
                decisionlevel ← level
            end if
        else
            if alle variabelen hebben waarden then
                return variabelen
            end if
            if keuzeliteral met niet-geprobeerde negatie op huidige level then
                kies negatie voor keuzeliteral
                decisionlevel ← decisionlevel + 1
            else
                kies nieuw keuzeliteral en ken waarde toe
                decisionlevel ← decisionlevel + 1
            end if
        end if
    end while
end function
```

Een voorbeeld van een SAT-solver die DPLL en Conflict-Driven Clause Learning gebruikt is Minisat.[Eén and Sörensson, 2003] Minisat neemt SAT-formules in de

conjunctieve normaalvorm als invoer. Minisat is ontworpen om gemakkelijk uitbreidbaar te zijn. MinisatID is een uitbreiding van deze SAT-solver en neemt ECNF als invoer. Dit is een uitbreiding van CNF, waar we naast clauses ook definities van de vorm $head \leftarrow body$ hebben, waarbij *body* een uitbreiding zoals aggregaten of vergelijkingen over gehele getallen bevat.

MinisatID werkt event-gebaseerd. Iedere toekenning van een waarde aan een variabele, door propagatie of door een keuze, is een event. Ook het veranderen van het beslissingsniveau, door backtracken van de solver of een nieuwe keuze te maken, is een event. Propagators kunnen zich abonneren op specifieke events. De SAT-solver roept de propagator op als één van de events voorkomt. De propagator probeert in deze oproep zoveel mogelijk extra informatie te propageren als mogelijk, zowel in de propagator zelf als extern, door een waarde toe te kennen aan een variabele. De propagator heeft hiervoor de beschikking over de huidige toekenning van alle variabelen in de solver. Als de propagator na een event niet meer satisfiable is, kan deze explanation clauses genereren en zal MinisatID backtracken, net zoals voor gewone conflicten.

MinisatID heeft propagators voor inductieve definities [Mariën et al., 2008], aggregaten [De Cat and Denecker, 2010], aritmetica en andere uitbreidingen. Door deze manier van werken combineert MinisatID Conflict-Driven Clause learning en extra propagators tot één solver waarin verschillende uitbreidingen langs elkaar bestaan.

2.5 De satisfiability checking taak

In het vervolg van deze thesis gaan we de SAT-solver MinisatID uitbreiden. Om een beter idee te hebben hoe de in- en uitvoer van deze SAT-solver er uit ziet, geven we allereerst een beschrijving van de in- en uitvoer van de SAT-solver MinisatID.

De in- en uitvoer van MinisatID is als volgt:

1. Invoer:
   a) Een verzameling van variabelen V
   b) Een ECNF-theorie T over (een subset van) de variabelen V
   c) Een verzameling literals A die we assumpties noemen, over de variabelen V
   d) Een verzameling variabelen D, over de variabelen V, waar de SAT-solver beslissingen over mag maken
2. Uitvoer:
   a) Ofwel: Een model, een toekenning van waarden *true* of $false$ aan iedere variabele uit D
   b) Ofwel: UNKNOWN
   c) Ofwel: UNSATISFIABLE, waarbij we een subset van assumpties teruggeven die verantwoordelijk zijn voor de unsatisfiability

We kunnen de assumpties A zien als een uitbreiding van de theorie T, waarbij we voor iedere assumptie een unit clause toevoegen.

We geven een model terug als er een toekenning van waarden aan elke variabele in D bestaat zodat elke clause in de ECNF-theorie en elke andere constraint satisfiable is onder de gegeven toekenning. Dit betekent dat er voor iedere clause in T minstens één literal waar is en voor alle andere constraints geldt dat de toekenning van waarden aan variabelen in D voldoet aan de constraints in de ECNF-theorie.

We geven UNSATISFIABLE terug als er voor alle mogelijke toekenningen van waarden aan variabelen in D een clause of constraint in de theorie T unsatisfiable is. We geven UNKNOWN terug als nog niet alle clauses en constraints gegarandeerd satisfiable zijn met een toekenning van waarden over de variabelen in D. Een clause is gegarandeerd satisfiable als er minstens één literal *true* is. Het resultaat kan enkel UNKNOWN zijn als $D \cup A$ niet gelijk is aan V.

Een SAT-solver zal zoals eerder gezegd aan iedere variabele een waarde proberen toe te kennen, zodat er aan de theorie, gegeven door de verschillende clauses, voldaan is. Als de solver een model teruggeeft, zal er voor iedere clause uit de ECNF-theorie T minstens één literal moeten zijn die er voor zorgt dat de clause waar is. Voor iedere propagator moet gelden dat de toekenning van waarden aan variabelen niet in conflict is met de constraint die de propagator representeert. Een propagator mag literals propageren die een gevolg zijn van een combinatie van de huidige toekenning van waarden en de constraints vanuit de propagator.

2.6 Conclusie

In dit hoofdstuk hebben we gezien wat een kennisbanksysteem is. We hebben meer in het specifiek gekeken naar het kennisbanksysteem IDP. Ook hebben we kort Clingo bestudeerd, een ander kennisbanksysteem. Aangezien het IDP kennisbanksysteem een SAT-solver gebruikt, die we in het vervolg van de thesis gaan uitbreiden, hebben we ook de werking van de SAT-solver MinisatID bekeken.

Hoofdstuk 3

Epistemische logica en zijn toepassingen

In dit hoofdstuk bespreken we epistemische logica, de belangrijkste drijfveer achter de uitbreidingen van MinisatID die we in deze thesis maken. We bespreken wat epistemische logica is, welke toepassingen deze logica heeft en hoe we dit kunnen gebruiken. Een groot deel van dit hoofdstuk is gebaseerd op "Epistemic Logic and it's Applications: Tutorial Notes"[Davis and Morgenstern, 1983].

3.1 Epistemische logica

Epistemische logica [Davis and Morgenstern, 1983] is een uitbreiding op klassieke logica, waarin we kunnen redeneren over het bezitten of niet bezitten van kennis. In klassieke logica kunnen we enkel redeneren met de kennis die aanwezig is. In epistemische logica kunnen we ook redeneren over het wel of niet bezitten van kennis en kan het wel of niet zeker weten van bepaalde kennis implicaties hebben op andere kennis. In klassieke logica kunnen we bijvoorbeeld schrijven: "De robot mag enkel vooruit rijden als er recht voor hem geen ravijn is". Het is het echter niet mogelijk om de volgende zin op natuurlijke wijze te schrijven: "De robot mag enkel vooruit rijden als hij *weet* dat er recht voor hem geen ravijn is". In epistemische logica kunnen we dat laatste voorbeeld wel verwoorden. In het eerste voorbeeldje gaan we uit van volledige kennis, in het tweede voorbeeld houden we rekening met de mogelijkheid van onvolledige kennis, zoals in het geval van onvolledige of conflicterende vloerplannen.

We kunnen klassieke logica uitbreiden met een epistemische modale operator K(know): als ϕ een zin is, dan is $K(\phi)$ ook een zin. Met deze operator K geven we aan dat we *weten* dat een bepaalde zin waar is. Een dergelijke zin evalueren we in een logische theorie, een verzameling van zinnen. Een formule $K(\phi)$ is waar als er geen enkel model is dat voldoet aan de logische theorie, maar niet aan de formule ϕ.

In de meest eenvoudige vorm van epistemische logica gaan we uit van de kennis van de kennisbank van één agent, een autonome eenheid. We kunnen dit echter gemakkelijk generaliseren naar gedistribueerde epistemische logica, waarin we kunnen redeneren over de kennis van meerdere agenten die met elkaar moeten samenwerken

om een bepaalde taak te kunnen afwerken. Iedere agent heeft kennis over bepaalde zaken en weet ook welke kennis andere agenten hebben. Door kennis te delen, kunnen agenten nieuwe kennis verwerven. In het voorbeeld van de robot kunnen we een tweede mogelijke agent inschakelen, een satelliet die vanuit de ruimte de weg kan verkennen. We kunnen in deze gedistribueerde epistemische logica dus zinnen verwoorden zoals "De robot weet of er een ravijn is recht voor hem, wanneer de satelliet in een baan rond de planeet hem de plattegrond heeft doorgestuurd." en "De robot weet dat de satelliet weet of er een ravijn is recht voor hem."

Geordende epistemische logica, een subset van epistemische logica, is formeel gedefinieerd als volgt:

Definitie 2: "Een geordende epistemische theorie over een vocabularium Σ is een paar $\langle \mathcal{T}, \leq \rangle$ van een eindige set theorieën $\mathcal{T}$ en een partiële orde $\leq$ op $\mathcal{T}$ zodat iedere $T \in \mathcal{T}$ een theorie is in de taal $\mathcal{L}_T$, die inductief gedefinieerd is als eerste-orde logica met één extra regel: Als ϕ een formule is in $\mathcal{L}_{T'}$ en $T' < T$, dan is $K_{T'}(\phi)$ een element van de taal $\mathcal{L}_T$" [Vlaeminck et al., 2012a]

In geordende epistemische logica [Vlaeminck et al., 2012a] leggen we een extra beperking op het gebruik van de modale operator. We creëren een hiërarchie van logische theorieën, waarbij zinnen in een bepaalde theorie niet kunnen spreken over kennis in diezelfde theorie, enkel over kennis in theorieën op een lager niveau. Deze subset van epistemische logica is hierdoor makkelijk voor te stellen als een uitbreiding van eerste-orde logica.

De modale operator $K_{T_i}(\phi)$ betekent "De theorie T_1 weet formule ϕ". Ondanks deze beperking kunnen we de meeste zinnen alsnog verwoorden, zoals "De robot heeft een plattegrond, doorgestuurd door de satelliet. Door de informatie op deze plattegrond weet hij dat er geen ravijn is recht voor hem."

3.2 Toepassingen van epistemische logica

Met behulp van epistemische logica kunnen we problemen in diverse domeinen op natuurlijke wijze modelleren. In deze sectie beschrijven we een aantal voorbeelden, zoals planning en analyse van gedistribueerde systemen.

3.2.1 Planning

Planningsproblemen zijn problemen waarin we een sequentie van acties zoeken waarmee we een begintoestand kunnen omvormen tot een eindtoestand. Een voorbeeld hiervan is een robot die telkens een stap richting het noorden, zuiden, oosten of westen moet zetten om een doel te bereiken. Een sequentie van stappen waarmee de robot het doel bereikt is een geldig plan. Ook het zoeken van een zo kort mogelijke route voor vrachtwagens die meerdere ladingen moeten afleveren is een voorbeeld van een planningsprobleem.

Een planningsprobleem bestaat uit verschillende onderdelen: een planningsdomein dat algemene informatie bevat over het domein waarin het planningsprobleem zich stelt, en een specifieke probleeminstantie. Het planningsdomein beschrijft met behulp van een aantal predicaten wat de mogelijke toestanden van de wereld zijn en verschillende acties die de toestand van de wereld kunnen veranderen. Sommige acties mogen we enkel uitvoeren als de toestand van de wereld aan bepaalde voorwaarden voldoet. De probleeminstantie beschrijft de begin- en eindsituatie van een specifiek planningsprobleem. De beginsituatie is meestal volledig gedefinieerd, de eindsituatie is meestal partieel gedefinieerd.

Een conformant planningsprobleem is een veralgemening van een dergelijk planningsprobleem waarin we te maken hebben met niet-deterministische componenten, zoals een niet-deterministische begintoestand of acties die meerdere mogelijke uitkomsten hebben.[Vlaeminck et al., 2012a] Een geldig plan voor een conformant planningsprobleem is een plan dat het doel bereikt, voor alle mogelijke gevallen. Een voorbeeld van een conformant planningsprobleem is een robot die zijn doel moet bereiken, maar op bepaalde plaatsen kan er een ravijn zijn. Hij weet niet van alle locaties of er een ravijn is of niet. De robot moet ondanks deze onvolledige informatie voorkomen dat hij in een ravijn valt.

Conformante planningsproblemen kunnen we voorstellen door een modale operator toe te voegen aan bestaande modelleringen. De zin "K(het doel is bereikt)" beschrijft dat we vereisen dat we zeker zijn dat de doeltoestand bereikt is.

Een andere mogelijkheid binnen het planningsdomein is het vinden van een plan waarbij het bezitten van bepaalde kennis een voorwaarde is om andere acties te kunnen uitvoeren. Een voorbeeld is een robot die een sensor heeft om te detecteren of er een ravijn is op de locatie naast hem. Nadat de robot zijn sensor heeft gebruikt om zijn plattegrond te updaten, kan hij bepalen welke weg hij op zal gaan. Hierbij kunnen we gebruik maken van geordende epistemische logica: in iedere stap kunnen we spreken over de kennis die we in de vorige stap hadden en de kennis die we extra verkregen hebben in de laatst genomen actie. [Vlaeminck et al., 2012b]

3.2.2 Analyse van gedistribueerde systemen

Gedistribueerde systemen zijn systemen die bestaan uit meerdere losse componenten die met elkaar samenwerken om één service aan te bieden. Iedere component heeft zijn eigen taak en heeft slechts beperkte informatie.

Gedistribueerde systemen zijn op een natuurlijke manier te modelleren in epistemische logica, waarin we uitgaan van meerdere agenten. We kunnen spreken over wat iedere component weet, welke kennis bepaalde componenten nog niet hebben maar wel nodig hebben om het probleem op te lossen, wat componenten weten over wat andere componenten weten, wat de gedistribueerde kennis is van alle componenten samen en wat algemeen bekende kennis is (die bijvoorbeeld in het systeem is ingebouwd). Op die manier kunnen we verifiëren of het gedistribueerde systeem wel effectief zijn taak altijd kan uitvoeren, en zo niet, waarom dit niet zo is.

Als we terugkijken naar het voorbeeld van de robot die samenwerkt met een satelliet, zien we dat ook dit een gedistribueerd systeem is, waarin de robot en de

satelliet informatie moeten delen om samen de volledige planeet te kunnen verkennen. Als er meerdere robots op de planeet rijden, kunnen deze samenwerken om te voorkomen dat ze verschillende sectoren van de planeet dubbel verkennen.

3.3 Conclusie

In dit hoofdstuk is epistemische logica voorgesteld, een krachtige taal waarin we kunnen redeneren met het bezit van kennis. We hebben ook een aantal toepassingen besproken, zoals planning en de analyse van gedistribueerde systemen. Deze toepassingen geven aan dat de epistemische operator de IDP taal krachtiger zal maken en dat we hiermee eenvoudiger modelleringen mee kunnen maken.

Hoofdstuk 4

QBF Solvers

In dit hoofdstuk bespreken we gekwantificeerde booleaanse formules (QBF). Eerst leggen we uit wat QBF is. Daarna bestuderen we een aantal technieken om satisfiability van QBF-formules te checken.

4.1 QBF

In sectie 2.4 hebben we het satisfiability probleem voor propositionele logica geïntroduceerd. Gekwantificeerde booleaanse formules (QBF) zijn een uitbreiding van propositionele logica, waarbij we existentiële en universele kwantoren toevoegen. QBF is gedefinieerd als volgt:

Definitie 3: Zinnen in QBF:

- Een propositioneel atoom p is een zin.
- Als ϕ en ψ zinnen zijn, dan zijn de volgende expressies ook zinnen:
  - $\neg\phi$
  - $\phi \wedge \psi$
  - $\phi \vee \psi$
  - $\phi \implies \psi$
  - $\phi \iff \psi$
  - $\exists p_1 \dots p_n : \phi$ waarbij $p_1 \dots p_n$ propositionele atomen zijn
  - $\forall p_1 \dots p_n : \phi$ waarbij $p_1 \dots p_n$ propositionele atomen zijn

In SAT-problemen is een probleem satisfiable als we minstens één oplossing vinden voor een propositionele formule. Een voorbeeld van een propositionele formule is:

$$(\text{links} \vee \text{rechts}) \wedge (\neg\text{links} \vee \neg\text{rechts}) \tag{4.1}$$

We kunnen deze formule ook in QBF formuleren, door hier een existentiële kwantor voor te zetten:

$$\exists \text{ links rechts} : (\text{links} \vee \text{rechts}) \wedge (\neg\text{links} \vee \neg\text{rechts}) \tag{4.2}$$

In QBF kunnen we naast de existentiële kwantor ook de universele kwantor gebruiken. Hierdoor kunnen we zinnen schrijven zoals:

$$\exists \text{ links rechts} : \forall \text{ boven onder} : (\text{links} \vee \text{rechts}) \wedge (\neg\text{links} \vee \neg\text{rechts}) \wedge (\text{boven} \vee \text{onder} \vee \text{links}) \tag{4.3}$$

Hierin zoeken we niet alleen een waarde voor “links” en “rechts”, maar ook of voor alle mogelijke waarden van “boven” en “onder” de theorie, bestaande uit drie clauses, satisfiable blijft. De kwantoren maken QBF dus een krachtigere taal dan propositionele logica.

De twee QBF-formules hierboven zijn beiden in QBF prenex normaalvorm, waarin alle kwantoren vooraan staan. De theorie, bestaande uit de clauses, heet in deze voorstelling ook wel de matrix. De QBF normaalvorm is meestal niet de meest optimale voorstelling. Door QBF-formules te herschrijven, komen we soms op een compactere en optimalere voorstelling.

We kunnen willekeurige QBF-formules omzetten naar de QBF prenex normaalvorm op een gelijkaardige manier als we willekeurige propositionele logica-formules kunnen omzetten naar CNF. We maken hierbij gebruik van de herschrijfregels in sectie 2.4, uitgebreid met een aantal extra regels, waarmee we kwantoren naar buiten brengen.

4.2 QBF Solvers

Aangezien QBF gelijkaardig is aan SAT, is het mogelijk om dit op te lossen met een gelijkaardig algoritme. SAT-problemen kunnen we oplossen met een algoritme gebaseerd op het DPLL-algoritme. We kunnen dit DPLL-algoritme generaliseren naar QBF, waarbij we de volgorde van de gekozen variabelen vastleggen op basis van de kwantoren. Als we een existentiële kwantor tegenkomen, mogen we hierover backtracken, als we een universele kwantor tegenkomen moet de formule waar zijn voor alle toekenningen. [Cadoli et al., 2002]

De QBF-solver Quaffle lost QBF-formules op met behulp van Conflict Driven Clause Learning, aangepast aan QBF.[Zhang and Malik, 2002] Quaffle is gebaseerd op een QBF-solver die DPLL als onderliggende beslissingsprocedure gebruikt. Het DPLL-algoritme voor QBF gebruikt, net zoals in het DPLL-algoritme voor SAT, unitpropagatie en beslissingen. Waar we in Conflict Driven Learning voor SAT de volgorde van de variabelen volledig zelf kunnen bepalen, zal voor QBF de volgorde vastgelegd zijn door de QBF-formule zelf. Voor de QBF-formule in vergelijking 4.3 zal een QBF-solver eerst een waarde voor “links” en “rechts” moeten kiezen, voordat “boven” en “onder” een waarde kunnen krijgen.

Een belangrijke stap in het oplossen van SAT en QBF-problemen, is het kunnen uitleggen van conflicten. In de SAT-solver MinisatID genereren we explanation clauses door middel van resolutie. In de QBF-solver Quaffle gebeurt dit door long distance resolutie, een veralgemening van resolutie voor QBF-formules.

Een mogelijke extra stap in het oplossen van QBF-formules is door deze formule eerst te preprocessen, waarbij we triviale gevallen van (un)satisfiability kunnen herkennen of de formule kunnen herschrijven naar een voorstelling die beter geschikt is voor de solver in kwestie.

Een aantal preprocessing technieken zijn beschreven in [Cadoli et al., 2002]. Een aantal gevallen van triviale (un)satisfiability worden hier herkend en de QBF-formule wordt herschreven om satisfiability eenvoudiger te kunnen checken.

De QBF-preprocessor Bloqqer [Biere et al., 2011] herkent geblokkeerde clauses, clauses die geen invloed hebben op de satisfiability van de QBF-formule. Dergelijke clauses kunnen we dus verwijderen uit een QBF-formule zonder de satisfiability aan te passen.

Zoals eerder gezegd is de prenex normalvorm niet de meest optimale voorstelling voor QBF-formules. We kunnen het aantal kwantoren in de QBF-formule verminderen door te werken met kwantificatie-bomen [Benedetti, 2005]. Een voorbeeld van een kwantificatie-boom is de QBF-formule

$$\exists \text{ onder} : (\forall \text{ links}(\text{onder} \vee \text{links})) \wedge (\forall \text{ rechts} : (\text{onder} \vee \text{rechts})) \tag{4.4}$$

We reduceren hier het bereik van kwantoren van een QBF-formule zo dat iedere clause enkel in het bereik van het strikt noodzakelijk aantal kwantoren valt.

4.3 Conclusie

In dit hoofdstuk hebben we gezien wat QBF-formules zijn. We hebben gezien hoe we satisfiability van QBF-formules kunnen bepalen en we hebben kort een aantal preprocessing technieken bekeken.

Hoofdstuk 5

Probleemstelling

Zoals duidelijk is geworden in vorige hoofdstukken, is epistemische logica een expressieve en krachtige taal. Geordende epistemische logica, een subset daarvan, is eenvoudig als uitbreiding op eerste-orde logica voor te stellen. Aangezien de IDP taal een uitbreiding is van eerste-orde logica, lijkt het daarom een logische keuze om ons te beperken tot deze geordende epistemische logica. Zeker door epistemische logica te combineren met de taalconcepten in de IDP taal kunnen we veel problemen eenvoudiger beschrijven. Het is dus interessant om te zien hoe geordende epistemische logica en de IDP taal te combineren te zijn en hoe we dit in het IDP-systeem kunnen inbouwen.

Om epistemische logica te kunnen ondersteunen in IDP, zijn er meerdere mogelijkheden. Een eerste mogelijkheid is om een vertaling van epistemische logica naar de bestaande taalconcepten in de IDP taal te maken. We kunnen echter ook de onderliggende architectuur uit van het IDP systeem uitbreiden om geordende epistemische logica op een natuurlijke manier te ondersteunen. In deze thesis hebben we ervoor gekozen om de onderliggende architectuur uit te breiden. Deze onderliggende architectuur bestaat, zoals eerder gezegd, uit een grounder en een solver.

Er was al langere tijd vraag naar het kunnen nesten van solvers in MinisatID. Hierdoor kunnen we problemen opdelen in kleinere stukken en elk van deze delen apart oplossen. Deze manier bevordert ook hergebruik van theorieën.

Door het nesten van solvers kunnen we ook QBF gaan oplossen met MinisatID. Aangezien epistemische logica op een natuurlijke manier naar QBF te vertalen is, lijkt deze techniek een goede manier om geordende epistemische logica te kunnen ondersteunen.

In deze thesis richten we ons specifiek op het uitbreiden van de solver. In de eerste plaats moeten we de invoertaal van de solver expressiever maken. We breiden deze invoertaal uit zodat we de IDP taal uitgebreid met geordende epistemische logica op een natuurlijke manier kunnen omzetten naar deze uitgebreide invoertaal. Hierin is het belangrijk dat de uitbreiding aan ECNF en het algoritme om deze uitgebreide ECNF op te lossen orthogonaal staat op andere taalconcepten in ECNF. Het doel is dus een solver te maken die de bestaande en toekomstige taalconcepten in ECNF kan combineren met QBF.

Nadat we de uitbreiding aan ECNF hebben ingevoerd, bestuderen we hoe we deze uitgebreide taal kunnen oplossen in de solver en welke voorstelling we hiervoor gebruiken. Eerst beschrijven we hoe we in de context van geneste ECNF clauses kunnen leren. Deze clauses kunnen we gebruiken om een algoritme uit te werken, dat geen kennis moet hebben van andere uitbreidingen van IDP. Dit algoritme is gebaseerd op lazy clause generation[Ohrimenko et al., 2007], waarin een propagator enkel een clause genereert op het moment dat we nieuwe informatie hebben en dus niet direct alle informatie in een propagator uitschrijven. Daarna vergelijken we een aantal varianten van dit algoritme, waarin we op verschillende momenten clauses leren.

Door QBF toe te voegen aan de invoertaal van MinisatID, kunnen we meer dan enkel epistemische logica ondersteunen. Ook andere logica's die we eenvoudiger naar geneste ECNF kunnen vertalen, kunnen hiervan profiteren.

Hoofdstuk 6

Een uitbreiding van ECNF voor geneste theorieën

In dit hoofdstuk introduceren we geneste ECNF, de taal waarmee we MinisatID uitbreiden om geordende epistemische logica te kunnen ondersteunen. Allereerst leggen we uit hoe we epistemische logica kunnen voorstellen met gekwantificeerde booleaanse formules. Daarna stellen we geneste ECNF voor, waarin we ECNF uitbreiden met QBF.

6.1 Uitdrukken van epistemische logica

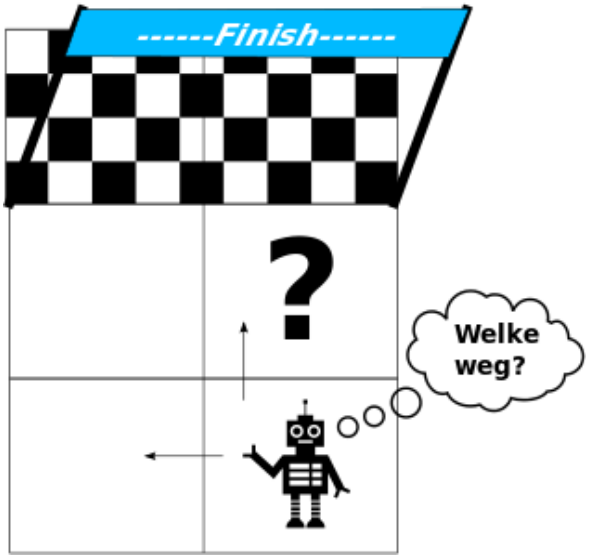
In hoofdstuk 4 hebben we gezien wat QBF is en hoe we deze QBF-formules kunnen oplossen. Met behulp van QBF kunnen we eenvoudig geordende epistemische logica uitdrukken. Een operator $K_{T_i}(\phi)$, waarbij de theorie T_i een vocabularium heeft met de variabelen $x_1 \dots x_n$, kunnen we vertalen met de zin $\forall x_1 \dots x_n : T_i \implies \phi$ (voor alle modellen van de theorie T_i moet ook de zin ϕ waar zijn). Op die manier kunnen we alle kennisoperatoren vervangen, wat resulteert in een QBF-zin.

We passen dit toe op een specifiek voorbeeld van een robot die moet navigeren op een terrein dat niet volledig verkend is. We stellen dit voorbeeld eerst grafisch voor, zoals te zien is in figuur 6.1.

De robot heeft in dit voorbeeld telkens twee mogelijke acties: naar voren bewegen of een zijwaartse beweging maken. We stellen "naar voren bewegen" voor door *true* en "een zijwaartse beweging maken" voor door $false$, voor de 3 stappen die de robot moet maken. Dit voorbeeld kunnen we in epistemische logica voorstellen als volgt:

$$\exists stap1\ stap2\ stap3 : (\neg K_{plattegrond}(\neg ravijn) \implies \neg stap1) \wedge ((stap1 \wedge stap2) \vee (stap1 \wedge stap3) \vee (stap2 \wedge stap3)) \quad (6.1)$$

Hierin beschrijven we dat als de robot niet weet dat er geen ravijn is op het vakje recht voor hem, hij niet vooruit mag gaan in de eerste stap. De robot heeft geen enkele kennis op zijn plattegrond over het ravijn, de theorie *plattegrond* is dus



Figuur 6.1: Het voorbeeld van de robot. De robot moet de finish bereiken, maar weet niet of het vakje recht voor hem een ravijn is of niet

leeg. De robot moet minstens 2 stappen vooruit zetten om de finish te bereiken. We kunnen dit herschrijven in QBF als volgt:

$$\exists stap1\ stap2\ stap3 : \forall ravijn : (\neg ravijn \implies \neg stap1) \wedge ((stap1 \wedge stap2) \vee (stap1 \wedge stap3) \vee (stap2 \wedge stap3)) \quad (6.2)$$

Zoals te zien kunnen we een formule in geordende epistemische logica dus eenvoudig vertalen naar QBF. We zijn dus op zoek naar een uitbreiding van ECNF om QBF-formules te kunnen schrijven. In de volgende sectie beschrijven we de semantiek van deze uitbreiding.

6.2 Semantiek van de uitbreiding

Onze taal, geneste ECNF, is een uitbreiding van de propositionele logica uit definitie 1 uitgebreid met definities. We breiden deze taal uit met universele ($\forall$) en existentiële ($\exists$) kwantoren. ECNF gebruikt definities om uitbreidingen zoals inductieve definities, aggregaten en aritmetica voor te stellen. In geneste ECNF voegen we een extra niet-recursieve definitie toe voor universele en existentiële kwantoren.

Definitie 4: Zinnen in geneste ECNF:

- Een propositioneel atoom p is een zin.
- Een definitie $head \leftarrow body$ is een zin.

- Als ϕ en ψ zinnen zijn, dan zijn de volgende expressies ook zinnen:
  - $\neg\phi$
  - $\phi \wedge \psi$
  - $\phi \vee \psi$
  - $\phi \implies \psi$
  - $\phi \iff \psi$
- Als *head* een propositioneel atoom is, ϕ een zin is en $x_1 \dots x_n$ zijn symbolen, dan zijn de volgende expressies ook zinnen:
  - $head \leftarrow \exists x_1 \dots x_n : \phi$
  - $head \leftarrow \forall x_1 \dots x_n : \phi$

Nu we deze taal gedefinieerd hebben, moeten we hier ook een semantiek aan toekennen. We laten de definities $head \leftarrow body$ buiten beschouwing. Een zin in geneste propositionele logica heeft enkel een betekenis onder een bepaald vocabularium in een bepaalde interpretatie. We beginnen met een initieel vocabularium V. Met behulp van existentiële en universele kwantoren kunnen we ons vocabularium uitbreiden en kunnen de subexpressies meer symbolen gebruiken. Een interpretatie geeft aan ieder symbool een waarde $true$ of $false$. We formaliseren dit in de volgende definitie:

Definitie 5: Formele semantiek van geneste CNF:

- $I \vDash \phi$ als en slechts als $I, V \vDash \phi$.
- $I, V \vDash p$ als en slechts als $p \in V$ en $I(p) = true$.
- $I, V \vDash \neg\phi$ als en slechts als $I, V \nvDash \phi$.
- $I, V \vDash \phi \wedge \psi$ als en slechts als $I, V \vDash \phi$ en $I, V \vDash \psi$.
- $I, V \vDash \phi \vee \psi$ als en slechts als $I, V \vDash \phi$ of $I, V \vDash \psi$.
- $I, V \vDash \phi \implies \psi$ als en slechts als $I, V \vDash \phi$ impliceert dat $I, V \vDash \psi$.
- $I, V \vDash \phi \iff \psi$ als en slechts als $I, V \vDash \phi \implies \psi$ en $I, V \vDash \psi \implies \phi$.
- $I, V \vDash head \leftarrow \exists x_1 \dots x_n : \phi$ als en slechts als voor $i = 1 \dots n$ geldt dat $x_i \notin V$ en als en slechts als $I, V \vDash head$, er een interpretatie $I' = I \cup \{x_1 \rightarrow bool \dots x_n \rightarrow bool\}$ bestaat zodat $I', V' \vDash \phi$ met $V' = V \cup \{x_1 \dots x_n\}$.
- $I, V \vDash head \leftarrow \forall x_1 \dots x_n : \phi$ als en slechts als voor $i = 1 \dots n$ geldt dat $x_i \notin V$ en als en slechts als $I, V \vDash head$, voor alle interpretaties $I' = I \cup \{x_1 \rightarrow bool \dots x_n \rightarrow bool\}$ geldt dat $I', V' \vDash \phi$ met $V' = V \cup \{x_1 \dots x_n\}$.

Een formule ϕ is satisfiable als en slechts als er een interpretatie I bestaat zodat $I \vDash \phi$.

Propositionele logica is equivalent aan de subset van deze taal waar we geen existentiële of universele kwantoren in onze zin hebben staan en geen definities hebben.

In het vervolg van deze thesis gaan we enkel uit van een beperkte subset van deze taal. We gebruiken hiervoor de volgende herschrijfregels, die satisfiability bewaren:

- $\phi \iff \psi$ vormen we om naar $(\phi \implies \psi) \wedge (\psi \implies \phi)$
- $\phi \implies \psi$ vormen we om naar $\neg\phi \vee \psi$
- $\forall x_1 \dots x_n : \phi$ vormen we om naar $\neg\exists x_1 \dots x_n : \neg\phi$

Met behulp van distributiviteit en de wetten van De Morgan kunnen we deze formules verder omvormen naar een normaalvorm die lijkt op de conjunctieve normaalvorm, waarbij er enkel clauses of definities met existentiële kwantoren overblijven op ieder niveau. De formule

$$(a \vee b \vee c) \wedge (\neg sub) \wedge (sub \leftarrow \exists x\ y\ z : (a \vee x) \wedge (a \vee y)) \tag{6.3}$$

is een voorbeeld van een formule in normaalvorm. In het vervolg van de thesis zullen we een dergelijke formule vaak schrijven als

$$(a \vee b \vee c) \wedge (\neg\exists x\ y\ z : (a \vee x) \wedge (a \vee y)) \tag{6.4}$$

om de leesbaarheid te verhogen.

6.3 Conclusie

In dit hoofdstuk is geneste ECNF voorgesteld, de uitbreiding aan ECNF om met epistemische logica te kunnen omgaan. Allereerst is de taal en zijn semantiek voorgesteld. Tot slot is uitgelegd hoe we epistemische logica naar deze uitbreiding kunnen vertalen. Dit hebben we ook toegepast op een voorbeeld.

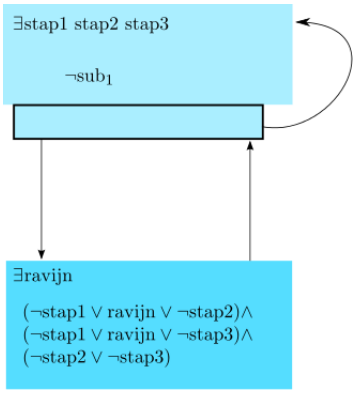
Hoofdstuk 7

Modelexpansie voor geneste ECNF

In het vorige hoofdstuk hebben we geneste ECNF geïntroduceerd. In dit hoofdstuk bespreken we hoe we deze geneste ECNF voorstellen in de SAT-solver MinisatID, aan welke voorwaarden een algoritme moet voldoen dat deze geneste ECNF kan oplossen in de context van MinisatID, en stellen we een specifiek algoritme voor dat deze geneste ECNF kan oplossen.

7.1 Omzetting van geneste ECNF in MinisatID

Om modelexpansie te kunnen doen op geneste ECNF, moeten we een representatie vinden voor geneste ECNF om dit te kunnen voorstellen in de solver MinisatID. We zetten geneste ECNF om naar een solver-subsolver structuur. Voor het voorbeeld gegeven in sectie 6.1 stellen we dit voor zoals in figuur 7.1.



FIGUUR 7.1: De solver-subsolver structuur uitgewerkt voor het voorbeeld met de robot

Voor iedere existentiële kwantor maken we een aparte subsolver aan. Voor iedere subsolver voegen we een literal toe aan de supersolver, die aangeeft of de subsolver satisfiable is of niet. Deze literal noemen we een subsolverliteral. Voor een zin omgezet vanuit QBF verwachten we dat alle subsolvers niet satisfiable zijn (door de omzetting van "$\forall$" naar "$\neg\exists\neg$"), maar het algoritme is te veralgemenen voor zinnen waarin de subsolver wel satisfiable moet zijn, of waarin nog niet bekend is of de subsolver satisfiable is of niet. Hierdoor kunnen we zinnen schrijven zoals "Als de subsolver satisfiable is, dan moet de zin ϕ ook waar zijn".

Iedere subtheorie heeft een aantal gedeelde symbolen, waarvan de supersolver de waarde zal bepalen en aantal eigen symbolen die niet voorkomen in de theorie van de supersolver.

7.2 Invoer en Uitvoer algoritme

Voordat we een specifiek algoritme kunnen bespreken, moeten we eerst bepalen wat geldige invoer en uitvoer voor een algoritme voor geneste ECNF is.

Als invoer voor het algoritme hebben we één supertheorie, die zich in de supersolver bevindt, met n subtheorieën, waarbij iedere subtheorie zich in een subsolver bevindt. Elke subsolver beschouwen we als een black box, een orakel. Iedere theorie kan subtheorieën bevatten, maar ook andere uitbreidingen zoals aggregaten of inductieve definities.

De solver en subsolver communiceren met elkaar door middel van een interface. De interface geeft de waarden van de gedeelde variabelen, die bepaald zijn door de supersolver, door naar de subsolver. Deze gedeelde variabelen krijgen hun waarden door middel van assumpties. Dit zijn keuzes die de solver niet zelf maakt, maar van buiten krijgt opgelegd.

Deze interface roept ook de subsolver op wanneer dit nodig is. We kunnen de subsolver oproepen als volledige solver (deze geeft dan "satisfiable" of "unsatisfiable" terug), maar we kunnen deze ook een voorlopig resultaat opvragen door de subsolver enkel te laten propageren en geen keuzes te toe te staan.

De interface is ook verantwoordelijk voor de waarde van de subsolverliteral. Als de subsolverliteral een waarde heeft, zal de interface controleren of deze waarde overeenkomt met de uitkomst van de subsolver, als deze literal geen waarde heeft zal de interface de juiste waarde toekennen.

Door de subsolver op te roepen komen we informatie te weten over de relatie tussen de gedeelde variabelen en de uitkomst van de subsolver. Deze relatie kunnen we vastleggen door middel van explanation clauses. In de volgende sectie bespreken we welke explanation clauses we kunnen leren en welke voorwaarden we hiervoor nodig hebben.

7.3 Geldige geleerde clauses

In deze sectie gaan we uit van één supersolver S met één subsolver B. De supersolver heeft een theorie T_{super} over het vocabularium $V_{\text{super}} \cup \{\text{sub}\}$, waarbij "sub" de

subsolverliteral is voor de subtheorie T_{sub} van B. De subsolver beschikt over een eigen vocabularium $W = \{w_1 \dots w_n\}$ als uitbreiding op V_{super}, zodat $V_{\text{sub}} = V_{\text{super}} \cup W$. We kunnen T_{sup} verder specificeren als volgt: $T' \wedge (\text{sub} \leftarrow \exists w_1 \dots w_n : T_{\text{sub}})$.

Als interpretatie I een volledige interpretatie is over alle symbolen uit V_{super} en de subtheorie T_{sub} is satisfiable met de interpretatie I als set van assumpties, dan is

$$I \implies \text{sub} \tag{7.1}$$

een geldig geleerde clause. Als de subtheorie T_{sub} unsatisfiable is onder de interpretatie I, dan is

$$I \implies \neg\text{sub} \tag{7.2}$$

een geldig geleerde clause.

In het geval dat de theorie T_{sub} unsatisfiable is, kunnen we in bepaalde gevallen een slimmere clause leren. De SAT-solver MinisatID heeft een procedure $unsatCore()$ die we kunnen oproepen op een solver. Deze procedure zoekt een (sub)set van assumpties die de theorie van deze solver unsatisfiable maakt. Met behulp van deze procedure kunnen we dus mogelijk een kleinere clause teruggeven. Als de procedure $unsatCore()$ een set van assumpties A' teruggeeft, kunnen we de clause

$$A' \implies \neg\text{sub} \tag{7.3}$$

leren, waarbij de set A' een subset is van I.

De voorgaande gevallen gaan er vanuit dat we een volledige interpretatie hebben voor de theorie T_{super}. In bepaalde gevallen kunnen we echter al een clause leren met een partiële interpretatie.

Als interpretatie I voor een subset van symbolen uit V_{super} een waarde geeft en we deze waarden als assumpties zetten in de subsolver, hebben we drie mogelijke uitkomsten: $satisfiable$, $unsatisfiable$ of $unknown$. We krijgen satisfiable als uitkomst als de subsolver B gegarandeerd satisfiable is onder de gegeven interpretatie, zonder symbolen uit V_{super} te kiezen of te propageren. Dit houdt in dat er voor iedere clause minstens één literal waar moet zijn en voor alle constraints moet gelden dat deze onder alle uitbreidingen van de partiële interpretatie I voldaan zijn. Als de subsolver gegarandeerd satisfiable is, kunnen we de clause

$$I \implies \text{sub} \tag{7.4}$$

leren, zoals we eerder ook in de context van een volledige interpretatie hebben gezien.

Het kan ook zijn dat de subsolver B unsatisfiable is onder de gegeven interpretatie. Dit geval kunnen we opsplitsen in twee gevallen: ofwel is er minstens één clause unsatisfiable zonder dat de subsolver een keuze maakt, ofwel is er minstens één clause unsatisfiable in alle mogelijke keuzes van de subsolver voor de symbolen van de subsolver V_{sub}. In het eerste geval is propagatie na het zetten van de assumpties voldoende, in het tweede geval moeten we, ondanks onvolledige informatie, veel berekenen. Als de subsolver unsatisfiable is onder de partiële interpretatie I kunnen we de clause

$$I \implies \neg\text{sub} \tag{7.5}$$

leren. Ook hier kunnen we eventueel gebruik maken van de $unsatCore()$ procedure.

Als we geen gegarandeerde (un)satisfiability hebben, is er nog niet voldoende informatie in de interpretatie I om satisfiability van de subtheorie te kunnen bepalen. We kunnen dus geen clause leren, als de subsolver geen keuzes of propagaties mag maken op de gedeelde symbolen V_{super}. Als we deze restrictie laten vallen, kunnen we mogelijk extra clauses leren.

Als we een partiële interpretatie I geven aan subsolver B en deze subsolver zonder keuzes te maken een literal v of $\neg v$ propageert, waarbij $v \in V_{\text{super}}$, kunnen we de clause

$$I \wedge \text{sub} \implies v \tag{7.6}$$

leren.

7.4 Lazy algoritme

Nu we hebben gezien welke geldige clauses we kunnen leren, gebruiken we dit om een volledig algoritme te beschrijven. In de eerste plaats beschrijven we een lazy algoritme, dat zo laat mogelijk de subsolver oproept.

Als eerste stap gebruiken we het standaard CDCL algoritme om een waarde te vinden voor alle symbolen in V_{super}. Als we een model kunnen vinden, zullen we voor iedere subtheorie ook een geldig model proberen vinden. We geven het model voor T_{super} via assumpties mee aan de subsolver. We laten de subsolver volledig solven en krijgen daarom enkel $true$ of $false$ als antwoord. Met dit antwoord kunnen we een clause leren, zoals in vergelijking 7.1 en vergelijking 7.2.

We kunnen dit algoritme verbeteren door in het geval unsatisfiability een clause te leren zoals in vergelijking 7.3.

We passen dit toe op een voorbeeld:

$$\begin{aligned}\exists stap1\ stap2\ stap3 : \neg\exists ravijn : (\neg ravijn \vee \neg stap1 \vee \neg stap2) \wedge \\ (\neg ravijn \vee \neg stap1 \vee \neg stap3) \wedge (\neg stap2 \vee \neg stap3)\end{aligned} \tag{7.7}$$

In de eerste stap zoeken we een toekenning van waarden voor $stap1$, $stap2$ en $stap3$. We kiezen er hier voor om aan alle variabelen $false$ toe te kennen. In de formules hieronder geven we $false$ aan met $\overline{v}$, terwijl we $true$ aangeven met $\underline{v}$.

$$\begin{aligned}\exists \overline{stap1}\ \overline{stap2}\ \overline{stap3} : \neg\exists ravijn : (\neg ravijn \vee \underline{\neg\overline{stap1}} \vee \underline{\neg\overline{stap2}}) \wedge \\ (\neg ravijn \vee \underline{\neg\overline{stap1}} \vee \underline{\neg\overline{stap3}}) \wedge (\underline{\neg\overline{stap2}} \vee \underline{\neg\overline{stap3}})\end{aligned} \tag{7.8}$$

Zoals te zien is de toekenning in de subsolver satisfiable, aangezien iedere clause satisfiable is. We leren de clause $\neg stap1 \wedge \neg stap2 \wedge \neg stap3 \implies \text{sub}$.

Om aan te tonen wanneer we de $unsatCore()$ procedure kunnen gebruiken, bestuderen we een gelijkaardig voorbeeld:

$$\exists stap1\ stap2\ stap3 : \exists ravijn : (\neg ravijn \vee \neg stap1 \vee \neg stap2) \wedge (\neg ravijn \vee \neg stap1 \vee \neg stap3) \wedge (\neg stap2 \vee \neg stap3) \quad (7.9)$$

De supersolver kiest in dit geval voor $stap1$, $stap2$ en $stap3$ de waarde $true$.

$$\exists \underline{stap1}\ \underline{stap2}\ \underline{stap3} : \exists ravijn : (\neg ravijn \vee \overline{\neg \underline{stap1}} \vee \overline{\neg \underline{stap2}}) \wedge (\neg ravijn \vee \overline{\neg \underline{stap1}} \vee \overline{\neg \underline{stap3}}) \wedge (\overline{\neg \underline{stap2}} \vee \overline{\neg \underline{stap3}}) \quad (7.10)$$

In dit geval is de subsolver unsatisfiable. We kunnen hier de clause $stap2 \wedge stap3 \implies \neg$ sub leren.

7.5 Eager algoritme

In het lazy algoritme roepen we de subsolvers pas op aan het einde van het zoekproces. Als we de subsolvers eerder oproepen, kunnen we soms al eerder satisfiability bepalen en dus al eerder clauses leren.

We roepen de subsolver op met partiële interpretatie I op na iedere propagatie of beslissing op een symbool uit V_{super} in de supersolver S. We laten de subsolver B enkel propageren, zonder beslissingen te maken. Als we gegarandeerde satisfiability hebben, leren we een clause zoals in vergelijking 7.4. Op dit moment kan MinisatID nog geen gegarandeerde satisfiability teruggeven, dit geval zal dan ook niet voorkomen in de huidige implementatie. Als de subsolver unsatisfiable is, leren we een clause zoals in vergelijking 7.5.

We passen dit toe op het eerder gegeven voorbeeld:

$$\exists stap1\ stap2\ stap3 : \neg\exists ravijn : (\neg ravijn \vee \neg stap1 \vee \neg stap2) \wedge (\neg ravijn \vee \neg stap1 \vee \neg stap3) \wedge (\neg stap2 \vee \neg stap3) \quad (7.11)$$

We laten de supersolver één toekenning doen. Deze kent $true$ toe aan $stap3$:

$$\exists stap1\ stap2\ \underline{stap3} : \neg\exists ravijn : (\neg ravijn \vee \neg stap1 \vee \neg stap2) \wedge (\neg ravijn \vee \neg stap1 \vee \overline{\neg \underline{stap3}}) \wedge (\neg stap2 \vee \overline{\neg \underline{stap3}}) \quad (7.12)$$

De subsolver propageert $\neg stap2$, maar kan hierna geen verdere propagaties doen en geeft unknown terug. De supersolver kiest hierna $true$ voor de variabele $stap2$:

$$\exists stap1\ \underline{stap2}\ \underline{stap3} : \neg\exists ravijn : (\neg ravijn \vee \neg stap1 \vee \neg \underline{stap2}) \wedge (\neg ravijn \vee \neg stap1 \vee \overline{\neg \underline{stap3}}) \wedge (\overline{\neg \underline{stap2}} \vee \overline{\neg \underline{stap3}}) \quad (7.13)$$

De subsolver geeft nu unsatisfiable terug. We kunnen hier de clause $stap2 \wedge stap3 \implies \neg$ sub leren.

7.6 Eager algoritme met propagatie

In het eager algoritme kan de subsolver propageren over symbolen uit V_{super}, maar we doen niets met deze propagaties. Zoals we eerder hebben gezien kunnen we ook in dit geval clauses leren. We roepen de subsolver op met partiële interpretatie I zoals in het voorgaande algoritme en laten deze propageren. Als de subsolver B een literal van de solver S propageert, leren we de clause zoals in vergelijking 7.6.

We passen dit nogmaals toe op het eerder gegeven voorbeeld:

$$\exists stap1\ stap2\ stap3 : \neg\exists ravijn : (\neg ravijn \vee \neg stap1 \vee \neg stap2) \wedge (\neg ravijn \vee \neg stap1 \vee \neg stap3) \wedge (\neg stap2 \vee \neg stap3) \quad (7.14)$$

We laten de supersolver zoals eerder één toekenning doen. Deze kent *true* toe aan $stap3$:

$$\exists stap1\ stap2\ \underline{stap3} : \neg\exists ravijn : (\neg ravijn \vee \neg stap1 \vee \neg stap2) \wedge (\neg ravijn \vee \neg stap1 \vee \overline{\neg \underline{stap3}}) \wedge (\neg stap2 \vee \overline{\neg \underline{stap3}}) \quad (7.15)$$

De subsolver zal $\neg stap2$ propageren. We kunnen hier de clause $stap3 \wedge \text{sub} \implies \neg stap2$ leren.

7.7 Experimentele evaluatie

We hebben de algoritmes hierboven uitgevoerd op een aantal testcases in QBF. Al deze QBF-formules waren in prenex normaalvorm, wat zoals eerder gezegd niet de meest optimale voorstelling is. Deze experimenten toonden aan dat het huidige algoritme nog niet concurrerend is ten opzichte van bestaande QBF-solvers wat betreft snelheid. De trend was dat het lazy algoritme meer testcases kon oplossen dan de eager algoritmes. Het lijkt er dus op dat het eerder oproepen van de subsolvers in het geval van QBF-formules in prenex normaalvorm niet voldoende extra informatie geeft om te compenseren voor de extra kost hiervan.

De MinisatID solver is op dit moment de enige solver die QBF en inductieve definities, aggregaten en aritmetica combineert. Het was dus niet mogelijk om de performantie van MinisatID te vergelijken met andere solvers op dit vlak.

7.8 Conclusie

In dit hoofdstuk hebben we een algoritme uitgewerkt om geneste ECNF op te kunnen lossen. We hebben allereerst besproken welke clauses we kunnen leren. We hebben daarna een algoritme geformuleerd dat modelexpansie doet op geneste ECNF met behulp van deze geleerde clauses. We hebben dit algoritme uitgebreid om al eerder in het zoekproces clauses te leren.

Hoofdstuk 8

Besluit

In deze thesis hebben we allereerst het IDP kennisbanksysteem bestudeerd. Dit systeem heeft een aantal inferentiemethodes die we op kennis geschreven in de IDP taal kunnen uitvoeren. We hebben gezien dat dit kennisbanksysteem is opgebouwd uit een grounder en een solver. We hebben deze solver, MinisatID, in meer detail bekeken. Nadat we het IDP kennisbanksysteem hebben bekeken, hebben we gezien wat epistemische logica is, de logica die we willen toevoegen aan het IDP kennisbanksysteem. Voordat we de probleemstelling introduceerden, hebben we eerst gekwantificeerde booleaanse formules (QBF) en QBF-solvers bestudeerd.

In deze thesis hebben we een uitbreiding gedaan aan de SAT-solver MinisatID. We hebben de taal van deze solver, ECNF, uitgebreid met kwantoren, waardoor we gekwantificeerde booleaanse formules in ECNF kunnen schrijven. We hebben deze uitgebreide ECNF "geneste ECNF" genoemd. Deze uitbreiding zorgt ervoor dat we de IDP taal, uitgebreid met geordende epistemische logica, kunnen omzetten naar geneste ECNF. Deze geneste ECNF stellen we in MinisatID voor met behulp van subsolvers.

We hebben besproken in welke gevallen we een clause kunnen leren en hoe een geldig geleerde clause eruit ziet in elk van deze gevallen. We hebben eerst een lazy algoritme opgesteld dat gebruik maakt van deze geleerde clauses om geneste ECNF uit te voeren. Daarna hebben we op basis van dit lazy algoritme een eager algoritme voorgesteld dat eerder unsatisfiability kan detecteren. Ook dit eager algoritme hebben we uitgebreid tot een algoritme waar we propagaties in de subsolver detecteren en gebruiken om clauses te leren.

Het resultaat van deze uitbreiding is een unieke solver die inductieve definities, aggregaten en aritmetica kan combineren met gekwantificeerde booleaanse formules.

8.1 Toekomstig werk

In deze thesis hebben we enkel de solver MinisatID uitgebreid. Om epistemische logica volledig te ondersteunen op deze manier zal ook de IDP taal uitgebreid moeten worden en moet de grounder deze uitgebreide IDP kunnen vertalen naar geneste ECNF.

Zoals gezegd zijn er meerdere manieren om geordende epistemische logica toe te voegen in IDP. Het zou interessant zijn om ook de vertaling naar bestaande taalconcepten van IDP te bekijken. Wellicht dat de IDP taal voldoende expressief is om geordende epistemische logica op een efficiënte manier te vertalen naar de IDP taal zonder deze logica. Ook het vertalen van de IDP taal uitgebreid met geordende epistemische logica naar bestaande ECNF kan een interessante piste zijn.

Zoals besproken in hoofdstuk 4 zijn er diverse preprocessing technieken mogelijk voor QBF-formules. Een aantal van deze technieken kunnen, met eventuele kleine aanpassingen, ook eenvoudig toegepast worden op geneste ECNF. We kunnen deze preprocessing technieken toevoegen op meerdere manieren. We kunnen allereerst een preprocessor plaatsen voor MinisatID, die kan omgaan met geneste ECNF. We kunnen een aantal preprocessing technieken echter ook in MinisatID zelf implementeren. QBF preprocessing technieken zorgen in veel gevallen voor een grote snelheidswinst in bestaande QBF-solvers. De verwachting is dat deze technieken ook voordelig zijn voor geneste ECNF.

Bijlagen

Bijlage A

Poster

Op de volgende pagina is de poster terug te vinden, die gepresenteerd werd op de postermarkt van 7 mei 2015.

Orakels in IDP

Een implementatie voor epistemische logica

Student: Neline van Ginkel
Begeleider: Bart Bogaerts
Promotor: prof. Denecker

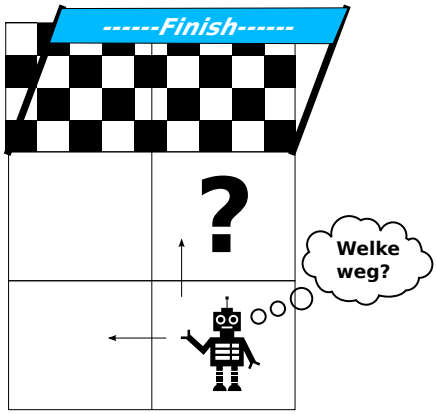
Achtergrond

Kennis in eerste-orde logica:
- "Als er een ravijn is recht voor de robot, mag hij niet vooruit gaan"

Kennis in epistemische logica:
- "De robot mag enkel rechtdoor gaan als hij weet dat er geen ravijn is recht voor hem"

Geordende Epistemische logica:
- Krachtige taal
- Kan rekening houden met het bezitten of niet bezitten van kennis
- Laat toe veel zaken natuurlijk te modelleren
- Heeft natuurlijke vertaling naar gekwantificeerde booleaanse formules (QBF)

QBF:
- SAT met kwantoren



QBF-solvers:
- Verschillende oplostechnieken mogelijk
- Verschillende soorten solvers, o.a. conflict-driven solvers
- Passen vaak extra technieken toe om efficiënter te kunnen solven
- Bestaande solvers beperkt tot "standaard" QBF

IDP:
- Kennisbanksysteem (verschillende taken mogelijk met dezelfde kennis)
- Expressieve invoertaal FO(.) ondersteunt o.a. inductieve definities, aggregaten, aritmetiek
- Nog geen ondersteuning voor epistemische logica

Doelstelling

Doel: Geordende Epistemische logica toevoegen aan het IDP kennisbanksysteem

IDP bestaat uit twee componenten:
- grounder vertaalt FO(.) naar lager-level ECNF theorie
- solver krijgt ECNF als invoer en probeert een model voor de theorie als oplossing te geven

In deze thesis focus op de solver:
- QBF toevoegen aan invoertaal van MinisatID (door deze uitbreiding kunnen ook hogere-orde logica en diverse andere logica's vertaald worden naar ECNF)
- Algoritme ontwerpen en implementeren waarmee QBF opgelost kan worden, dat orthogonaal staat op bestaande uitbreidingen (inductieve definities, aggregaten, aritmetiek)

Aanpak

Leidend voorbeeld:
Robot moet de finish bereiken, maar weet niet of er een ravijn recht voor hem is

Vertaling naar QBF:

$\exists stap1\ stap2\ stap3 : \forall ravijn : (stap1 \wedge \neg ravijn \wedge stap2) \vee (stap1 \wedge \neg ravijn \wedge stap3) \vee (stap2 \wedge stap3)$

Kies stappen vooruit(true)/zijwaarts(false)
Er is wel/geen ravijn op het vakje met "?"

$\exists stap1\ stap2\ stap3$
$\forall ravijn$
$(stap1 \wedge \neg ravijn \wedge stap2) \vee (stap1 \wedge \neg ravijn \wedge stap3) \vee (stap2 \wedge stap3)$

Vertaling naar geneste ECNF:
- universele kwantoren vervangen door existentiële kwantoren

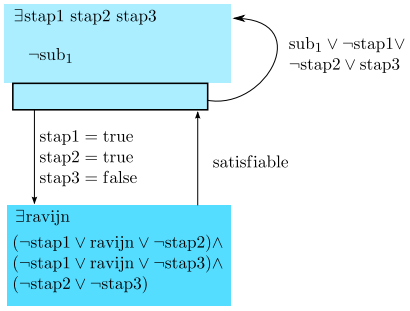
$\exists stap1\ stap2\ stap3 : \neg\exists ravijn : (\neg stap1 \vee ravijn \vee \neg stap2) \wedge (\neg stap1 \vee ravijn \vee \neg stap3) \wedge (\neg stap2 \vee \neg stap3)$

- subtheorieën expliciet maken

$\exists stap1\ stap2\ stap3 : \neg sub_1$

$sub_1 == \exists ravijn : (\neg stap1 \vee ravijn \vee \neg stap2) \wedge (\neg stap1 \vee ravijn \vee \neg stap3) \wedge (\neg stap2 \vee \neg stap3)$

- omzetten naar solver-subsolver structuur



Oplossen in solver-subsolver structuur

Supersolver: kiest waarden voor eigen symbolen
Interface -> subsolver: geeft waarden door
Subsolver: zoekt oplossing
Subsolver->interface: resultaat
Interface->supersolver: resultaat voor supersolver
Interface->supersolver: geleerde clause voor supersolver

Diverse keuzes blijven mogelijk in algoritme:

Wanneer wordt de subsolver gerund?
- run subsolver enkel als alle literals bekend zijn
- run subsolver zodra een literal bekend is, geef voorlopig resultaat terug

Met welke informatie worden er clauses geleerd?
- alle assumpties (literals bepaald door supersolver)
- subset van assumpties (extra rekenwerk om kleinere subset te krijgen)

Wanneer wordt er een clause geleerd?
- Als de solver volledig gesolved heeft
- Zodra de subsolver een literal van de supersolver propageert

Bijlage B

Wetenschappelijke paper

Op de volgende pagina's is de wetenschappelijke paper terug te vinden.

Oracles in IDP – an implementation for epistemic logic

Neline van Ginkel
KU Leuven
neline.vanginkel@student.kuleuven.be

Abstract

In this paper we propose an extension to the IDP knowledge base system to add epistemic logic, a logic of knowledge, to the IDP language. More specifically, we describe an extension to the SAT-solver MinisatID, in which we allow the nesting of SAT-solvers. We propose an extension to the input language ECNF, we describe properties that hold in this new setting and propose an algorithm to solve this nested ECNF.

This results in a unique solver that combines inductive definitions, aggregates and arithmetics with quantified boolean formulas.

1. Introduction

In classic logic we can reason with knowledge, but we cannot reason with the lack of certain knowledge. Epistemic logics [1] fill this gap in classic logic in order to reason with the existence or non-existence of knowledge as well. In classic logic we can state "The robot is not allowed to drive forward if there is ravine in front of him", but we cannot express "The robot is only allowed to drive forward if he *knows* there is no ravine in front of him."

It would be an interesting approach to extend the IDP knowledge base system[2] with epistemic logic. The input language for this system is the IDP language, an extension of first-order logic with constructs for inductive definitions, aggregates and arithmetics. The IDP knowledge base system consists of two parts: a grounder, that translates the IDP language to the lower-level language ECNF, and a solver, that finds a solution for problem stated in ECNF. In this paper we extend MinisatID, the SAT-solver underlying the IDP knowledge base system, with quantified boolean formulas in order to prepare support for epistemic logic. This extension is orthogonal on existing extensions of MinisatID.

2. Background

Epistemic logics are logics of knowledge, in which we can reason about the current state of information and gain and loss thereof. [1] These logics can be used in diverse AI applications, such as conformant planning, planning with knowledge acquiring steps and sensing [3], but also in more general Computer Science applications, such as the analysis of distributed systems.

Ordered epistemic logic [4] is an epistemic logic that can be considered a direct extension of first-order logic.

Definition 1. *"An ordered epistemic theory over vocabulary Σ is a pair $\langle \mathscr{T}, \leq \rangle$ of a finite set $\mathscr{T}$ of theories and a partial order $\leq$ on $\mathscr{T}$, such that each $T \in \mathscr{T}$ is a theory in the language $\mathscr{L}_T$, which is inductively defined as first order logic with one extra formula-building rule: If ϕ is a formula in $\mathscr{L}_{T'}$ and $T' < T$, then $K_{T'}(\phi)$ is in $\mathscr{L}_T$." [4]*

A SAT-solver solves satisfiability-problems for propositional logic. Given a SAT-problem, we search an assignments of values to a set of variables for a formula in propositional logic. This assignment can be found with an algorithm based on the backtrack-based DPLL-algorithm, extended with Conflict-Driven Clause Learning [5].

MinisatID[6] is a SAT-solver based on Conflict-Driven Clause Learning. The input language for is ECNF, Extended Conjunctive Normal Form, which extends SAT with constructs such as inductive definitions, aggregates and integer arithmetics.

Quantified Boolean Formulas (QBF) are a generalisation of the SAT problem, in which each variable is quantified by an existential or universal quantifier. We can translate ordered epistemic logic to quantified boolean formulas. We consider a theory T_{sub} over a vocabulary $V = v_1 \dots v_n$. A formula "$K_{T_i}(\phi)$" can be translated to "$\forall v_1 \dots v_n : T_{\text{sub}} \implies \phi$".

We can solve QBF using a QBF-solver, such as Quaffle[7]. In this solver Conflict-Driven Clause Learning is adapted for quantified boolean formulas. While

we can choose the order of assignment arbitrary in SAT, we need to follow the order of quantifiers in QBF. For each universal quantifier, we need to check both paths, instead of just one path. Quaffle uses long-distance resolution in order to explain conflicts with conflict-clauses.

3. Semantics of nested ECNF

We extend ECNF with an additional non-recursive definition, similar to quantified boolean formulas (QBF).

Definition 2. *Nested ECNF:*

- *A propositional atom p is a sentence.*
- *A definition $head \leftarrow body$ is a sentence. Extensions such as aggregates and arithmetics are represented with definitions.*
- *If ϕ and ψ are sentences, then the following expressions are also sentences:*
  - $\neg\phi$
  - $\phi \wedge \psi$
  - $\phi \vee \psi$
  - $\phi \implies \psi$
  - $\phi \iff \psi$
- *If $head$ is a propositional atom, ϕ is a sentence and $x_1 \dots x_n$ are symbols, then the following expressions are also sentences:*
  - $head \leftarrow \exists x_1 \dots x_n : \phi$
  - $head \leftarrow \forall x_1 \dots x_n : \phi$

Now that we have defined nested ECNF, we also have to define its semantics.

Definition 3. *Semantics of nested ECNF. We omit the semantics for definitions.*

- $I \vDash \phi$ *if and only if* $I, V \vDash \phi$.
- $I, V \vDash p$ *if and only if* $p \in V$ *and* $I(p) = true$.
- $I, V \vDash \neg\phi$ *if and only if* $I, V \nvDash \phi$.
- $I, V \vDash \phi \wedge \psi$ *if and only if* $I, V \vDash \phi$ *and* $I, V \vDash \psi$.
- $I, V \vDash \phi \vee \psi$ *if and only if* $I, V \vDash \phi$ *or* $I, V \vDash \psi$.
- $I, V \vDash \phi \implies \psi$ *if and only if* $I, V \vDash \phi$ *implies that* $I, V \vDash \psi$.
- $I, V \vDash \phi \iff \psi$ *if and only if* $I, V \vDash \phi \implies \psi$ *and* $I, V \vDash \psi \implies \phi$.
- $I, V \vDash head \leftarrow \exists x_1 \dots x_n : \phi$ *if and only if for* $i = 1 \dots n$ *it holds that* $x_i \notin V$ *and if and only if* $I, V \vDash head$, *there exists an interpretation* $I' = I \cup \{x_1 \rightarrow bool \dots x_n \rightarrow bool\}$ *such that* $I', V' \vDash \phi$ *where* $V' = V \cup \{x_1 \dots x_n\}$.
- $I, V \vDash head \leftarrow \forall x_1 \dots x_n : \phi$ *if and only if for* $i = 1 \dots n$ *it holds that* $x_i \notin V$ *and if and only if* $I, V \vDash head$ *it holds that for all interpretations* $I' = I \cup \{x_1 \rightarrow bool \dots x_n \rightarrow bool\}$ *it holds that* $I', V' \vDash \phi$ *where* $V' = V \cup \{x_1 \dots x_n\}$.

A formula ϕ is satisfiable if and only if there exists an interpretation I such that $I \vDash \phi$.

We rewrite nested ECNF to a variant of the Conjunctive Normal Form, consisting of a conjunction of clauses and definitions. We rewrite all universal quantifiers to existential quantifiers, using the rewrite rule $\forall x = \neg\exists\neg$. The formula

$$\exists \text{ left right} : \forall \text{ up} : (\text{left} \vee \text{right}) \wedge (\neg\text{left} \vee \neg\text{right}) \wedge (\text{left} \vee \text{up}) \quad (1)$$

is rewritten in nested ENCF as

$$\exists \text{ left right } p : (\text{left} \vee \text{right}) \wedge (\neg\text{left} \vee \neg\text{right}) \wedge (\neg p) \wedge (p \leftarrow \exists \text{ up} : (\text{left} \vee \text{up})) \quad (2)$$

4. Solving nested ECNF

We represent the rule $head \leftarrow \exists x_1 \dots x_n$ in MinisatID as a separate propagator. This propagator forms the interface between a separate SAT-solver representing the body of the definition and the head of the rule, that represents the satisfiability of the subsolver.

We first describe valid learned clauses. Afterwards, we will describe several variants of an algorithm that uses these learned clauses.

4.1. Valid learned clauses

We describe several valid learned clauses in this context. We assume a single supersolver S and a subsolver B. The supersolver S has a theory T_{super} over the vocabulary $V_{\text{super}} \cup \{\text{head}\}$, with "head" the literal representing the theory T_{sub} of B. The subsolver has a separate vocabulary $W = \{w_1 \dots w_n\}$ in extension to V_{super}, such that $V_{\text{sub}} = V_{\text{super}} \cup W$. We can specify T_{sup} as following: $T' \wedge (\text{sub} \leftarrow \exists w_1 \dots w_n : T_{\text{head}})$.

If interpretation I is a full interpretation of vocabulary V_{super} and the theory T_{sub} is satisfiable with the set of assumptions I, then

$$I \implies \text{sub} \quad (3)$$

is valid learned clause. If the theory T_{sub} is unsatisfiable with the set of assumptions I, then

$$I \implies \neg\text{sub} \tag{4}$$

is a valid learned clause. If A' is a subset of I such that the theory T_{sub} is unsatisfiable with the set of assumptions A', then

$$A' \implies \neg\text{sub} \tag{5}$$

is also a valid learned clause. The SAT-solver MinisatID has a procedure $unsatCore()$ that returns a set of assumptions A' that satisfies the above condition.

If interpretation I is a interpretation for a subset of vocabulary V_{super}, there are three possible outcomes: the subsolver is satisfiable under any interpretation extending I, the subsolver is unsatisfiable under any interpretation extending I, or the result of the subsolver is still unknown. If the subsolver is satisfiable under any interpretation extending I, then

$$I \implies \text{sub} \tag{6}$$

is a valid learned clause. If the subsolver is unsatisfiable under any interpretation extending I, then

$$I \implies \neg\text{sub} \tag{7}$$

is a valid learned clause. If the satisfiability is still unknown, we cannot learn a clause in this way. If we allow the subsolver to propagate on symbols in V_{super}, we can possibly learn additional clauses. If I is a partial interpretation of symbols in V_{super}, and subsolver B propagates the value of a symbol $v \in V_{\text{super}}$ without making any choices, then

$$I \wedge \text{sub} \implies v \tag{8}$$

is a valid learned clause.

We use these valid learned clauses to formulate an algorithm that can solve nested ECNF.

4.2. Lazy algorithm

We first describe a lazy algorithm, that delays calling the subsolver as much as possible.

In the first step we use the standard Conflict-Driven Clause Learning algorithm to find a value for each symbol in V_{super}. If we find a model I for the theory T_{super}, we check if this model can be extended to a model for the theory T_{sub}. We set the model I as assumptions for the subsolver B. We run the subsolver as a usual SAT-solver and get the answer $satisfiable$ or $unsatisfiable$. If the subsolver is satisfiable, we learn a clause such as in equation 3, if the subsolver is unsatisfiable we learn a clause such as in equation 4.

We improve this algorithm by learning a clause such as in equation 5, with the $unsatCore()$ procedure.

4.3. Eager algorithm

In the lazy algorithm, we only call the subsolver as the last step of the search process. If we call the subsolver earlier in the process, we might be able to determine (un)satisfiability earlier and thus learn clauses faster.

In the first step we use the standard Conflict-Driven Clause Learning algorithm as before, but every time one of the symbols in V_{super} gets assigned a value, we call the subsolver. The subsolver is not allowed to make any choices and will thus only propagate. If the subsolver returns $satisfiable$, we learn a clause such as in equation 6. If the subsolver returns $unsatisfiable$, we learn a clause such as in equation 7. If the subsolver returns *unknown*, we cannot learn any clause. If all symbols in V_{super} have a value, we run the subsolver as a usual SAT-solver and only get $satisfiable$ or $unsatisfiable$ as an answer.

4.4. Eager algorithm with additional propagation

In the eager algorithm, the subsolver can propagate in every step, but the result of this propagation is ignored as long as the result of the solver is *unknown*. If the subsolver propagates a value $v \in V_{\text{super}}$ in the subsolver, we can learn a clause such as in equation 8.

5. Conclusion

In this paper we presented an extension to the MinisatID SAT-solver. We extended the ECNF language with an additional construct: definitions with quantifiers. This extension supports the representation of arbitrary quantified boolean formulas. We described several valid learned clauses in the setting of nested ECNF and described several variants of an algorithm to solve this nested ECNF.

6. Bibliography

References

[1] E. Davis and L. Morgenstern, "Epistemic logic and its applications: Tutorial notes," in *International Joint Conferences on Artificial Intelligence*, 1983, printed.

[2] S. D. Pooter, J. Wittocx, and M. Denecker, "A prototype of a knowledge-based programming environment," in *Applications of Declarative Programming and Knowledge Management - 19th International Conference, INAP 2011, and 25th Workshop on Logic Programming, WLP 2011, Vienna, Austria, September 28-30, 2011, Revised Selected Papers*, 2011, pp. 279–286.

[3] H. Vlaeminck, J. Vennekens, and M. Denecker, "A general representation and approximate inference algorithm for sensing actions," in *AI 2012: Advances in Artificial Intelligence - 25th Australasian Joint Conference, Sydney, Australia, December 4-7, 2012. Proceedings*, ser. Lecture Notes in Computer Science, M. Thielscher and D. Zhang, Eds., vol. 7691. Springer, 2012, pp. 543–554. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-35101-3

[4] H. Vlaeminck, J. Vennekens, M. Bruynooghe, and M. Denecker, "Ordered epistemic logic: Semantics, complexity and applications." in *KR*, 2012.

[5] J. P. M. Silva, I. Lynce, and S. Malik, "Conflict-driven clause learning SAT solvers," in *Handbook of Satisfiability*, ser. Frontiers in Artificial Intelligence and Applications, A. Biere, M. Heule, H. van Maaren, and T. Walsh, Eds. IOS Press, 2009, vol. 185, pp. 131–153. [Online]. Available: http://dx.doi.org/10.3233/978-1-58603-929-5-131

[6] B. De Cat, "Separating knowledge from computation: An fo (.) knowledge base system and its model expansion inference," Ph.D. dissertation, KU Leuven, Belgium, 2014.

[7] L. Zhang and S. Malik, "Conflict driven learning in a quantified boolean satisfiability solver," in *Proceedings of the 2002 IEEE/ACM International Conference on Computer-aided Design, ICCAD 2002, San Jose, California, USA, November 10-14, 2002*, 2002, pp. 442–449. [Online]. Available: http://doi.acm.org/10.1145/774572.774637

Bibliografie

[Benedetti, 2005] Benedetti, M. (2005). Quantifier trees for qbfs. In Bacchus, F. and Walsh, T., editors, *Theory and Applications of Satisfiability Testing, 8th International Conference, SAT 2005, St. Andrews, UK, June 19-23, 2005, Proceedings*, volume 3569 of *Lecture Notes in Computer Science*, pages 378–385. Springer.

[Biere et al., 2011] Biere, A., Lonsing, F., and Seidl, M. (2011). Blocked clause elimination for QBF. In Bjørner, N. and Sofronie-Stokkermans, V., editors, *Automated Deduction - CADE-23 - 23rd International Conference on Automated Deduction, Wroclaw, Poland, July 31 - August 5, 2011. Proceedings*, volume 6803 of *Lecture Notes in Computer Science*, pages 101–115. Springer.

[Bogaerts et al., 2014] Bogaerts, B., Jansen, J., Bruynooghe, M., de Cat, B., Vennekens, J., and Denecker, M. (2014). Simulating dynamic systems using linear time calculus theories. *TPLP*, 14(4-5):477–492.

[Cadoli et al., 2002] Cadoli, M., Schaerf, M., Giovanardi, A., and Giovanardi, M. (2002). An algorithm to evaluate quantified boolean formulae and its experimental evaluation. *J. Autom. Reasoning*, 28(2):101–142.

[Davis and Morgenstern, 1983] Davis, E. and Morgenstern, L. (1983). Epistemic logic and its applications: Tutorial notes. In *International Joint Conferences on Artificial Intelligence*. Printed.

[Davis et al., 1962] Davis, M., Logemann, G., and Loveland, D. W. (1962). A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397.

[De Cat, 2014] De Cat, B. (2014). *Separating Knowledge from Computation: An FO (.) Knowledge Base System and its Model Expansion Inference*. PhD thesis, KU Leuven, Belgium.

[De Cat and Denecker, 2010] De Cat, B. and Denecker, M. (2010). Dpll (agg): An efficient smt module for aggregates. In *LaSh 2010 Workshop*.

[Eén and Sörensson, 2003] Eén, N. and Sörensson, N. (2003). An extensible sat-solver. In *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, pages 502–518.

[Gebser et al., 2014] Gebser, M., Kaminski, R., Kaufmann, B., and Schaub, T. (2014). Clingo = ASP + control: Preliminary report. *CoRR*, abs/1405.3694.

[Mariën et al., 2006] Mariën, M., Wittocx, J., and Denecker, M. (2006). The idp framework for declarative problem solving. *Search and Logic: Answer Set Programming and SAT*, pages 19–34.

[Mariën et al., 2008] Mariën, M., Wittocx, J., Denecker, M., and Bruynooghe, M. (2008). SAT(ID): satisfiability of propositional logic extended with inductive definitions. In Büning, H. K. and Zhao, X., editors, *Theory and Applications of Satisfiability Testing - SAT 2008, 11th International Conference, SAT 2008, Guangzhou, China, May 12-15, 2008. Proceedings*, volume 4996 of *Lecture Notes in Computer Science*, pages 211–224. Springer.

[Ohrimenko et al., 2007] Ohrimenko, O., Stuckey, P. J., and Codish, M. (2007). Propagation = lazy clause generation. In Bessiere, C., editor, *Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings*, volume 4741 of *Lecture Notes in Computer Science*, pages 544–558. Springer.

[Pooter et al., 2011] Pooter, S. D., Wittocx, J., and Denecker, M. (2011). A prototype of a knowledge-based programming environment. In *Applications of Declarative Programming and Knowledge Management - 19th International Conference, INAP 2011, and 25th Workshop on Logic Programming, WLP 2011, Vienna, Austria, September 28-30, 2011, Revised Selected Papers*, pages 279–286.

[Silva et al., 2009] Silva, J. P. M., Lynce, I., and Malik, S. (2009). Conflict-driven clause learning SAT solvers. In Biere, A., Heule, M., van Maaren, H., and Walsh, T., editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 131–153. IOS Press.

[Vlaeminck et al., 2012a] Vlaeminck, H., Vennekens, J., Bruynooghe, M., and Denecker, M. (2012a). Ordered epistemic logic: Semantics, complexity and applications. In *KR*.

[Vlaeminck et al., 2012b] Vlaeminck, H., Vennekens, J., and Denecker, M. (2012b). A general representation and approximate inference algorithm for sensing actions. In Thielscher, M. and Zhang, D., editors, *AI 2012: Advances in Artificial Intelligence - 25th Australasian Joint Conference, Sydney, Australia, December 4-7, 2012. Proceedings*, volume 7691 of *Lecture Notes in Computer Science*, pages 543–554. Springer.

[Zhang and Malik, 2002] Zhang, L. and Malik, S. (2002). Conflict driven learning in a quantified boolean satisfiability solver. In *Proceedings of the 2002 IEEE/ACM International Conference on Computer-aided Design, ICCAD 2002, San Jose, California, USA, November 10-14, 2002*, pages 442–449.

Fiche masterproef

Student: Neline van Ginkel

Titel: Orakels in IDP – Een implementatie voor epistemische logica

Engelse titel: Oracles in IDP – An implementation for epistemic logic

UDC: 681.3

Korte inhoud:
Epistemische logica is een krachtige logica om te redeneren met het bezitten of niet bezitten van kennis. In deze thesis onderzoeken we hoe we een subset van deze logica, geordende epistemische logica kunnen toevoegen aan het IDP kennisbank systeem, een systeem waarin we kennis kunnen representeren en diverse inferentietaken kunnen uitvoeren op deze kennis. We ondersteunen deze geordende epistemische logica door gekwantificeerde booleaanse formules (QBF) toe te voegen aan de SAT-solver MinisatID. We stellen geneste ECNF voor, een uitbreiding van de invoertaal ECNF. We introduceren ook een algoritme dat deze geneste ECNF kan oplossen en bespreken hier verschillende varianten van. Het resultaat is een krachtige solver die gekwantificeerde booleaanse formules combineert met bestaande expressieve concepten zoals inductieve definities, aggregaten en aritmetica.

Thesis voorgedragen tot het behalen van de graad van Master of Science in de ingenieurswetenschappen: computerwetenschappen

Promotor: Prof. dr. M. Denecker

Assessoren: Bart Bogaerts
Prof. dr. ir. F. Piessens

Begeleider: Bart Bogaerts