KU LEUVEN

FACULTEIT INGENIEURSWETENSCHAPPEN

Automatisch vertalen van logigrammen naar een getypeerde logica

Jens Claes

Thesis voorgedragen tot het behalen van de graad van Master of Science in de ingenieurswetenschappen: computerwetenschappen, hoofdspecialisatie Artificiële intelligentie

Promotor:
Prof. dr. M. Denecker

Assessoren:
Prof. dr. M.-F. Moens
Dr. B. Bogaerts

Begeleiders:
Dr. B. Bogaerts
Ir. L. Janssens

Academiejaar 2016 – 2017

Inhoudsopgave

Samenvatting

Deze thesis evalueert en verbetert het semantische framework van Blackburn en Bos [5, 6]. Specifiek passen we dit framework toe op het vertalen van logigrammen naar logica. Hiervoor stellen we een set van lexicale categorieën en een grammatica op, specifiek voor logigrammen, op basis van tien logigrammen. We evalueren het framework op tien nieuwe logigrammen. Hierbij onderzoeken we of de nieuwe logigrammen, mits aanpassingen, uitdrukbaar zijn in de opgestelde grammatica en wat voor aanpassingen dan nodig zijn.

Verder breiden we het framework uit met types. Dankzij types kunnen grammaticaal correcte zinnen zonder betekenis, zoals “Het gras drinkt het zingende huis”, toch uitgesloten worden. Binnen deze thesis worden de types gebruikt om de verschillende domeinen van een logigram af te leiden. Daarnaast staan types ons toe om te vertalen naar een getypeerde logica.

Lijst van figuren en tabellen

Lijst van figuren

Lijst van tabellen

Lijst van afkortingen en symbolen

Afkortingen

CNL	Constructed Natural Language
DCG	Definite Clause Grammar
DRT	Discourse Representation Theory
DRS	Discourse Representation Structure
KBS	Knowledge Base System

Lexicale en Grammaticale categorieën

De verschillende soorten constituenten die kunnen voorkomen in een grammatica (terminologie uit de taalkunde). We gebruiken de Engelse afkortingen voor gelijkaardigheid met de literatuur.

s	Sentence (zin)
np	Noun Phrase (naamwoordgroep)
vp	Verb Phrase (verbale constituent)
ap	Adjective Phrase (adjectiefconstituent)
v	Verb (werkwoord)
iv	Intransitive Verb (onovergankelijk werkwoord)
tv	Transitive Verb (overgankelijk werkwoord)
av	Auxiliary Verb (hulpwerkwoord)
cop	Copular Verb (koppelwerkwoord)
pn	Proper Noun (eigennaam)
det	Determinator
noun	Noun (zelfstandig naamwoord)
n	(Modified) Noun (getransformeerd zelfstandig naamwoord)
prep	Preposition (voorzetsel)
pp	Prepositional Phrase (voorzetselconstituent)
number	Hoofdtelwoord
relpro	Relative pronoun (betrekkelijk voornaamwoord)
rc	Relative clause (betrekkelijke bijzin)
comp	Comparatief (bv. “lower than”)
some	Onbepaalde woorden (bv. “sometime”)
coord	Coordinator (voegwoord)

Inleiding

Logigrammen zijn een soort van puzzels waarbij de lezer een aantal zinnen voorgeschoteld krijgt. De zinnen bevatten een aantal domeinen (zoals nationaliteit, dier, kleur, ...) en een aantal domeinelementen (Noor, Brit, kat, hond, rood, blauw, ...). Ze beschrijven één bijectie tussen elk paar van domeinen. Het doel van de puzzel is het achterhalen van de waarde van de bijecties. M.a.w. welke domeinelementen bij elkaar horen. Bijvoorbeeld "De Noor woont in het blauwe huis en heeft een kat als huisdier". Elke puzzel heeft exact één oplossing.

De zinnen van logigrammen zijn redelijk gestructureerd waardoor het mogelijk is om ze automatisch om te vormen naar een formele representatie waarop inferenties uitgevoerd kunnen worden. Deze thesis onderzoekt hoe haalbaar deze automatische vertaling van logigrammen naar logica is. Dit dient als opstap naar meer algemene vertalingen van een natuurlijke taal naar een formele taal.

We bestuderen hiervoor het framework van Blackburn en Bos [5, 6]. Dit framework is gebaseerd op lambda-calculus en Frege's compositionaliteitsprincipe (de betekenis van een woordgroep is een combinatie van de betekenissen van de woorden of woordgroepen waaruit ze bestaat).

We geven eerst wat achtergrond die kan helpen bij het begrijpen van de thesis. Dan beschrijven we een aantal vertalingen van een natuurlijke taal naar een formele taal uit de literatuur. Daarna overlopen we het systeem dat we gebruiken om logigrammen automatisch op te lossen. Vervolgens stellen we het framework van Blackburn en Bos voor en leggen we uit hoe we dit framework kunnen gebruiken voor het vertalen van logigrammen naar logica. Eerst bespreken we het lexicon, vervolgens de grammatica. Daarna breiden we het framework uit met types en vervolgens illustreren we hoe we a.d.h.v. deze types een volledige specificatie kunnen opstellen. Ter evaluatie passen we dit hele framework toe op een aantal logigrammen uit Puzzle Baron's Logic Puzzles Volume 3 [20].

Motivatie

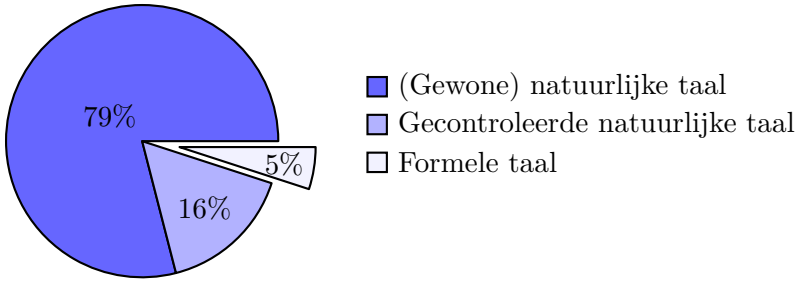
Vele bedrijfsprocessen worden geregeld door specificaties. Deze worden vaak geschreven in een natuurlijke taal, door een domein expert. Vervolgens worden deze specificaties vertaald naar uitvoerbare programma's. Bij deze vertaling kunnen er fouten insluipen. Bovendien zijn er vaak meerdere programma's die elk opnieuw de specificatie moeten implementeren. Zo ontstaan er niet alleen inconsistenties met de specificatie, maar ook tussen de verschillende programma's onderling. Ten slotte is het moeilijk om de specificatie achteraf nog aan te passen omdat alle programma's dan aangepast moeten worden.

Het antwoord van de academische wereld op deze problemen, is het *Knowledge Base*-paradigma. In dit paradigma staat een kennisbank centraal. Deze kennisbank bevat de kennis over de wereld en vormt een specificatie van hoe een systeem zich zou moeten gedragen. Deze kennisbank kan dan gebruikt worden in een *Knowledge Base System (KBS)*. Zo'n systeem biedt een aantal inferenties aan die toegepast kunnen worden op deze kennisbanken. De Cat et al. [10] geven het voorbeeld van een KBS dat gebruikt wordt om de vakken van een universiteit te beheren. De kennisbank in zo'n systeem bevat regels als "In elk lokaal en op elk moment mag er maximum één les plaatsvinden". Het systeem kan dan verschillende inferenties hierop uitvoeren. De Cat et al. geven o.a. de voorbeelden van *propagatie* (bijvoorbeeld in het individuele studieprogramma van een student automatisch de vakken toevoegen die een vereiste zijn om andere vakken te volgen die expliciet door de student werden gekozen), *model expansie* (bijvoorbeeld het opstellen van een volledig studieprogramma op basis van een partieel programma) en *bevraging* (bijvoorbeeld het opvragen van een uurrooster voor een specifieke student).

Op basis van de output van de inferenties kunnen de bedrijfsprocessen dan geregeld worden. Om de kennis voor te stellen, kan men gebruik maken van een formele taal (zoals eerste-orde-logica). Zo'n talen hebben een eenduidige semantiek. Er is dus maar één mogelijke manier waarop een zin geïnterpreteerd kan worden. Dit in tegenstelling tot natuurlijke talen waar zelfs simpele zinnen al snel ambigu zijn.

In het Knowledge Base-paradigma moet de specificatie in natuurlijke taal (geschreven door een domein expert) vertaald worden naar een vorm waar het KBS mee overweg kan: de formele specificatie. Deze specificatie wordt slechts eenmaal geschreven en vervolgens wordt ze gebruikt voor alle soorten inferenties. Daardoor is het niet meer mogelijk dat de programma's onderling inconsistent zijn. Ze gebruiken namelijk allemaal dezelfde formele specificatie. Bovendien is ook het aanpassen van de specificatie makkelijker. Enkel de specificatie moet aangepast worden, de programma's blijven hetzelfde.

Het probleem met deze aanpak is dat er nog steeds een vertaling moet gebeuren van natuurlijke taal naar een formele taal. De specificatie in natuurlijke taal wordt vaak opgesteld door een domein expert die niet vertrouwd is met formele talen. De formele specificatie wordt dan weer opgesteld door een KBS expert. Deze persoon kent formele



Figuur 1: Gebruik van natuurlijke taal in vereistenanalyse in 1999 (van figuur 5 in [15])

talen maar heeft een beperkte kennis van het domein. Door deze mismatch van expertise, sluipen er fouten in de vertaling. De domein expert kan namelijk de subtiliteiten van de formele taal niet lezen. Vice versa kent de KBS expert de subtiliteiten van het domein niet.

De vraag rijst dus of we een formele taal kunnen ontwerpen die toegankelijk is voor domein experten, rijk genoeg is voor praktische problemen en toepasbaar is binnen het KBS paradigma.

Deze thesis onderzoekt of een formele natuurlijke taal het antwoord is op die vraag. Hieronder verstaan we (een subset van) een natuurlijke taal met een formele, eenduidige semantiek. Talen die een subset zijn van een natuurlijke taal worden ook wel gecontroleerde natuurlijke talen of CNL's (naar het Engelse *Controlled Natural Language*) genoemd. Deze talen verschillen van hun gasttaal doordat ze een aantal zinsconstructies niet toelaten. Dit kan bijvoorbeeld de leesbaarheid van een taal verhogen. Voor dit onderzoek zijn deze talen interessant omdat ambigue constructies op die manier verboden kunnen worden. Bovendien is de grammatica van een CNL veel simpeler dan die van de volledige natuurlijke taal, waardoor het makkelijker is om er een parser voor te schrijven. De zinnen die toegestaan zijn in de CNL worden vaak beschreven in een set van *constructieregels*.

Naast constructieregels bevat een formele natuurlijke taal vaak ook interpretatieregels. Deze laatste bepalen hoe een zin die ambigu is in de gasttaal, geïnterpreteerd moet worden in de nieuwe taal. De moeilijkheid ligt erin om deze regels te beperken in aantal en in complexiteit, zodanig dat de geconstrueerde taal zo dicht mogelijk tegen de gasttaal aanleunt.

Het grote voordeel van een formele natuurlijke taal is dat natuurlijke taal al gebruikt wordt bij het opstellen van de specificatie. Een voorbeeld van een domein waar specificaties een grote rol spelen is vereistenanalyse. Hier zien we dat specificaties vaak in natuurlijke taal worden opgesteld. Zo toont figuur 1 het gebruik van natuurlijke taal in vereistenanalyse in 1999 [15]. Slechts 5 procent van de specificaties werd toen in een formele taal opgesteld. Al de rest werd in een natuurlijke taal geformuleerd. 16 procent werd zelfs al in een gecontroleerde natuurlijke taal opgesteld.

We voeren ons onderzoek naar zo'n formele natuurlijke taal uit binnen het domein van logigrammen. Ze kunnen namelijk aanzien worden als kleine specificaties. Bovendien kunnen ze uitgedrukt worden in relatief simpele logische zinnen. Ten slotte kan men makkelijk vele voorbeelden vinden van zo'n logigrammen. We zullen hierbij zelf geen grammatica opstellen maar een aantal grammaticale regels afleiden van bestaande logigrammen (uit Puzzle Baron's Logic Puzzles Volume 3 [20]).

Hoofdstuk 1

Achtergrond

Om deze thesis te begrijpen worden hier een aantal concepten uitgelegd die niet per se tot de achtergrondkennis behoren van de lezer. Definite Clause Grammars zijn een soort van grammatica die een aantal voordelen biedt voor het parsen van natuurlijke taal. Feature structures zorgen voor een beperking van het aantal grammaticale regels en verhogen de leesbaarheid ervan. Discourse Representation Theory is een theorie uit de taalkunde voor het vatten van de betekenis van taal. Het introduceert Discourse Representation Structures, een logica die dichter aanleunt bij de natuurlijke taal.

1.1 Definite Clause Grammars

Definite Clause Grammars [18] zijn grammatica's die expressiever zijn dan contextvrije grammatica's. Ze zitten vaak ingebakken in logische talen zoals prolog. Pereira et al. [18] geven 3 voorbeelden van hoe DCG's kunnen helpen bij het parsen van natuurlijke talen:

1. De woordvorm kan afhankelijk gemaakt worden van de context waarin deze verschijnt. Zo kan men eisen dat een werkwoord in de juiste vervoeging voorkomt.
2. Tijdens het parsen kan men een boom opbouwen die de semantiek van de zin moet vatten. Deze boom hoeft niet isomorf te zijn met de structuur van de grammatica.
3. Het is mogelijk om (prolog) code toe te voegen die extra restricties oplegt aan de grammatica.

1.1.1 Een eerste grammatica

Een voorbeeld van een DCG is grammatica 1.1.

```
1 s() --> np(), vp().
2 np() --> [ik].
3 np() --> [hem].
4 vp() --> v(), np().
5 v() --> [zie].
```

Grammatica 1.1: Een eerste grammatica

`s` is het startsymbool en staat voor `sentence`. `np` staat voor `noun phrase` (naamwoordgroep of nominale constituent), `vp` voor `verb phrase` (verbale constituent) en `v` voor `verb`. Deze grammatica zegt dat een zin bestaat uit een noun phrase gevold door een verb phrase. Een verb phrase is dan weer een werkwoord gevolgd door een noun phrase.

De zin *"ik zie hem"* is onderdeel van deze taal. Maar ook de zin *"ik zie ik"* is deel van de taal. Om dit op te lossen kunnen we argumenten meegeven aan de niet-terminaal `np`.

1.1.2 De woordvorm is afhankelijk van de context

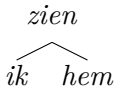
De verbeterde grammatica 1.2 houdt rekening met welke woordvorm kan voorkomen in welke context. De `nom` en `acc` slaan hier op de naamvallen `nominatief` en `accusatief`. Ze geven aan in welke functie de naamwoordgroepen gebruikt mogen worden binnen een zin. Namelijk als onderwerp of als lijdend voorwerp.

```
1 s() --> np(nom), vp().
2 np(nom) --> [ik].
3 np(acc) --> [hem].
4 vp() --> v(), np(acc).
5 v() --> [zie].
```

Grammatica 1.2: Een grammatica met argumenten

1.1.3 Een boom als resultaat

Verder is het ook mogelijk om een boom op te bouwen tijdens het parsen. Dit gebeurt in grammatica 1.3. Bij het parsen van *"ik zie hem"* krijgen we nu volgende boom als waarde voor het eerste argument van `s`.



Merk op dat deze boom de structuur van de parse tree niet hoeft te volgen. Het werkwoord wordt hier tot belangrijkste woord van de zin gebombardeerd.

```
1 s(Tree) --> np(NP, nom), vp(Tree, NP).
2 np(ik, nom) --> [ik].
3 np(hem, acc) --> [hem].
4 vp(Tree, Subject) --> v(Tree, Subject, Object), np(Object, acc).
5 v(zien(Subject, Object), Subject, Object) --> [zie].
```

Grammatica 1.3: Een grammatica die een boom opbouwt

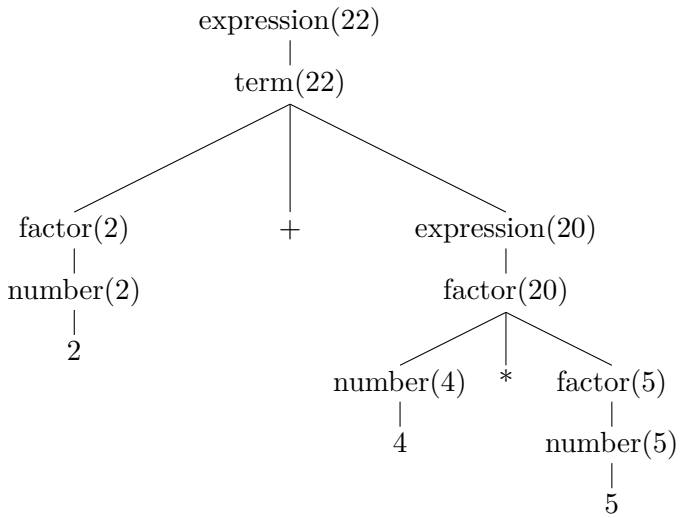
1.1.4 Prolog-code in de grammatica

Ten slotte is het mogelijk om prolog restricties te embedden in de grammatica door deze prolog goals tussen accolades te plaatsen.

```
1 expression(X) --> factor(X).
2 expression(X) --> term(X).
3
4 factor(X) --> numeral(X).
5 factor(X) --> numberal(A), [*], factor(B), {X is A * B}.
6 term(X) --> factor(A), [+], expression(B), {X is A + B}.
7
8 numeral(X) --> [X], {number(X)}.
```

Grammatica 1.4: Een grammatica met embedded prolog-code

Grammatica 1.4 kan simpele wiskunde expressies omvormen tot hun wiskundige waarde. Zo wordt `2 + 4 * 5` omgevormd tot `22` volgens volgende boom. Hierbij wordt de waarde van onder uit de boom naar boven toe gepropageerd via unificatie.



De prologcode in de accolades heeft twee functies. Enerzijds berekent die de waarde van een subexpressie zoals een factor of een term. Anderzijds beperkt de prologcode de grammatica. Een `numeral` bestaat uit 1 token maar enkel als dat token een getal is volgens prolog. Zo'n beperking in prolog kan ook gebruikt worden om uit de beperkingen van een contextvrije grammatica te treden.

1.1.5 Conclusie

Definite Clause Grammars zijn expressieve grammatica's die uitermate geschikt zijn voor het modelleren van de grammatica van een natuurlijke taal. In deze thesis zullen alle grammatica's dan ook gegeven worden in de vorm van een DCG.

1.2 Feature structures

Een feature structure is een term uit de taalkunde. Men kan zo'n structuur zien als *named arguments* voor een niet-terminaal symbool die gebruikt worden om een explosie aan grammaticale regels te voorkomen. Zo kan een `np` en `vp` een feature `getal` hebben dat aangeeft of de woordgroep in het enkelvoud of meervoud staat. De grammaticale regel voor een zin kan dan aangeven dat het onderwerp en werkwoord moeten overeenkomen in getal. Andere features geven bijvoorbeeld de naamval aan van een naamwoordgroep. Blackburn en Striegnitz [7] geven de volgende grammaticale regel als voorbeeld (hierbij staat `CAT` voor de grammaticale categorie van een woordgroep, vergelijkbaar met de naam van een niet-terminaal symbool):

$$\left[\begin{array}{ll} \text{CAT} & \text{s} \end{array}\right] \rightarrow \left[\begin{array}{ll} \text{CAT} & \text{np} \\ \text{NAAMVAL} & \text{nom} \\ \text{GETAL} & \boxed{1} \end{array}\right] \left[\begin{array}{ll} \text{CAT} & \text{vp} \\ \text{GETAL} & \boxed{1} \end{array}\right]$$

Deze regel zegt dat een zin bestaat uit een `np` gevolgd door een `vp`. De `np` moet de naamval `nom` (nominatief) hebben. Bovendien moet het getal van de `np` en de `vp` unificeren (de $\boxed{1}$ is een variabele).

Grammatica's die gebruik maken van feature structures, gebruiken altijd unificatie voor het samenvoegen van meerdere structuren. Niet alle features moeten namelijk altijd een waarde toegekend krijgen. Zo kan een eigennaam voorkomen als onderwerp en als lijdend voorwerp (en heeft dus geen waarde voor de feature `naamval`). Terwijl *"ik"* enkel als onderwerp kan voorkomen (en dus wel een waarde heeft voor die feature). Zoals in het

voorbeeld hierboven kan men door unificatie ook controleren of meerdere woordgroepen dezelfde waarden hebben voor een feature.

Zoals Shieber et al. [26] aanhalen, verschillen prolog termen van feature structures enkel in vorm[1]. Qua expressiviteit voegen feature structures echter niets toe aan DCG's. Zo is bovenstaande grammaticale regel op basis van feature structures equivalent met grammatica 1.5[2].

```
1 s() --> np([naamval:nom, getal:Getal]), vp([tijd:_, getal:Getal]).
```

Grammatica 1.5: Een DCG-grammatica equivalent aan de regel met feature structures

In deze thesis zullen we altijd bovenstaande notatie gebruiken om de argumenten van een DCG-grammatica te noteren. De argumenten zijn namelijk taalkundig verantwoord en kunnen aanzien worden als feature structures.

Feature structures (en dus ook argumenten in DCG's) zijn handig om de explosie van grammatica regels te voorkomen. Grammatica 1.6 (van Blackburn en Striegnitz [7]) is een voorbeeld van een grammatica zonder feature structures.

```
1  s() --> np_singular(), vp_singular().
2  s() --> np_plural(), vp_plural().
3  np() --> np_singular().
4  np() --> np_plural().
5  np_singular() --> det(), n_singular().
6  np_plural() --> det(), n_plural().
7  vp_singular() --> intransitive_verb_singular().
8  vp_singular() --> transitive_verb_singular(), np_singular().
9  vp_singular() --> transitive_verb_singular(), np_plural().
10 vp_plural() --> intransitive_verb_plural().
11 vp_plural() --> transitive_verb_plural(), np_singular().
12 vp_plural() --> transitive_verb_plural(), np_plural().
13 n_singular() --> [man].
14 ...
```

Grammatica 1.6: Een grammatica zonder feature structures (uit [7])

Hierbij staat de `n` voor zelfstandig naamwoord (van het Engelse `noun`) en `det` voor determinator. Deze grammatica can veel korter gemaakt worden door gebruik te maken van feature structures. Grammatica 1.7 (ook van Blackburn en Striegnitz [7]) is equivalent aan grammatica 1.6 maar maakt gebruikt van feature structures.

```
1 s() --> np([num:Num]), vp([num:Num]).
2 np([num:Num]) --> det(), n([num:Num]).
3 vp([num:Num]) --> intransitive_verb([num:Num]).
4 vp([num:Num]) --> transitive_verb([num:Num]), np([num:_]).
5 n([num:singular]) --> [man].
6 ...
```

Grammatica 1.7: Een equivalente grammatica met feature structures (ook uit [7])

[1] Zo speelt de volgorde in prolog wel een rol. Bovendien moet men in een DCG steeds alle argumenten vermelden, ook als ze ongebonden zijn.

[2] De volgorde van de features `naamval` en `getal` is belangrijk. De tijd van het werkwoord moet men vermelden, ook al maakt het niet uit voor deze regel.

Conclusie Door gebruik te maken van feature structures is de grammatica simpeler en leesbaarder. Bovendien hoeft het concept dat een zin bestaat uit een `np` gevolgd door een `vp` maar één keer te worden uitgedrukt. De feature structures zorgen voor de congruentie in getal van het onderwerp met het werkwoord.

1.3 Discourse Representation Theory

Voor het vertalen van natuurlijke taal naar logica zouden we graag gebruik maken van Frege's compositionaliteitsprincipe: De betekenis van een zin bestaat uit de combinatie van de betekenissen van de delen ervan. Als we eerste-orde-logica als doeltaal van onze vertaling nemen, komen we echter vrij snel in de problemen. Neem bijvoorbeeld de zin "If a man lives, he breathes". De vertaling hiervan in eerste-orde-logica is $\forall x \cdot man(x) \Rightarrow breathes(x)$. De vertaling van "a man lives" is echter $\exists x \cdot man(x)$, wat geen deel uit maakt van de betekenis van de hele zin. Blackburn en Bos [6] geven nog een aantal andere problemen met eerste-orde-logica als doeltaal. Zo is eerste-orde-logica ook niet handig voor het oplossen van anaforische referenties.

Ze suggereren Discourse Representation Structures als alternatief. Bos [8] definieert deze structuren als een lijst van *discourse referents* (woordgroepen waarnaar andere woordgroepen kunnen verwijzen) en een lijst van condities i.v.m. die referenties. Blackburn en Bos [6] geven o.a. een vertaling van deze DRS-structuren naar eerste-orde-logica. DRS-structuren hebben dus ook een formele betekenis. Zoals we verder zullen aantonen, zijn ze echter beter geschikt als doeltaal. Van hieruit kan dan verder vertaald worden naar eerste-orde-logica volgens de vertaling van Blackburn en Bos.

Wij hernemen hier hun vertaling naar eerste-orde-logica als semantiek voor deze structuren. Daarnaast geven we ook een inductieve definitie van zowel een DRS als een DRS-conditie.

Zij x_i variabelen en γ_i DRS-condities, dan is $\begin{array}{|l} \hline x_1, \ldots, x_n \\ \hline \gamma_1 \\ \ldots \\ \gamma_m \\ \hline \end{array}$ een DRS met als vertaling

$$\left(\begin{array}{|l} \hline x_1, \ldots, x_n \\ \hline \gamma_1 \\ \ldots \\ \gamma_m \\ \hline \end{array} \right)^{fo} = \exists x_1 \ldots \exists x_n \left((\gamma_1)^{fo} \wedge \ldots \wedge (\gamma_m)^{fo} \right)$$

Een lege DRS heeft als vertaling $\left(\begin{array}{|l} \hline \\ \hline \\ \hline \end{array} \right)^{fo} = true$.

Zij B_1 en B_2 allebei een DRS, dan is ook $\{B_1 \oplus B_2\}$ een DRS. De $\oplus$-connector kan men zien als de logische conjunctie van twee DRS-structuren. De vertaling bestaat uit het samenvoegen van de twee DRS-structuren:

$$\left(\left\{ \begin{array}{|l} \hline x_1, \ldots, x_k \\ \hline \gamma_1 \\ \ldots \\ \gamma_l \\ \hline \end{array} \oplus \begin{array}{|l} \hline y_1, \ldots, y_m \\ \hline \delta_1 \\ \ldots \\ \delta_n \\ \hline \end{array} \right\} \right)^{fo} = \left(\begin{array}{|l} \hline x_1, \ldots, x_k, y_1, \ldots, y_m \\ \hline \gamma_1 \\ \ldots \\ \gamma_l \\ \delta_1 \\ \ldots \\ \delta_n \\ \hline \end{array} \right)^{fo}$$

Dit zijn alle mogelijke DRS-structuren.

Zij R een predicaat met n argumenten, x_i een variabele, τ_i een term (uit eerste-orde-logica) en B_i een DRS, dan zijn $R(x_1, \ldots, x_n)$, $\tau_1 = \tau_2$, $\neg B$, $B_1 \vee B_2$ en $B_1 \Rightarrow B_2$ alle mogelijke DRS-condities. De vertaling naar eerste-orde-logica is:

$$\Big(R(x_1, ..., x_n)\Big)^{fo} = R(x_1, ..., x_n)$$

$$\Big(\tau_1 = \tau_2\Big)^{fo} = \Big(\tau_1 = \tau_2\Big)$$

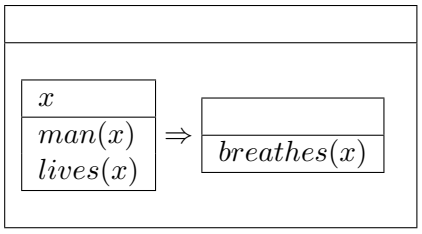
$$\Big(\neg B\Big)^{fo} = \neg\Big(B\Big)^{fo}$$

$$\Big(B_1 \vee B_2\Big)^{fo} = \Big(B_1\Big)^{fo} \vee \Big(B_2\Big)^{fo}$$

$$\left(\begin{array}{|l} \hline x_1, ..., x_n \\ \hline \gamma_1 \\ ... \\ \gamma_m \end{array} \Rightarrow B\right)^{fo} = \forall x_1 ... \forall x_n \left(\Big((\gamma_1)^{fo} \wedge ... \wedge (\gamma_m)^{fo}\Big) \Rightarrow \Big(B\Big)^{fo}\right)$$

Er zijn twee vertaalstappen van natuurlijke taal naar eerste-orde-logica. Eerst van natuurlijke taal naar DRS en vervolgens van DRS naar eerste-orde-logica. De laatste stap is niet compositioneel. De vertaling van de laatste DRS-conditie is dit namelijk niet. De variabelen van het hoofd van de implicatie zijn universeel gekwantificeerd i.p.v. existentieel.

De eerste stap is echter wel compositioneel. De vertaling van “If a man lives, he breathes” naar DRS is



De betekenis van het deel “a man lives” $\begin{array}{|l} \hline x \\ \hline man(x) \\ lives(x) \end{array}$ maakt deel uit van de betekenis van de hele zin.

1.4 Conclusie

Definite Clause Grammars vormen een expressieve grammatica die natuurlijke taal in al haar facetten makkelijk kan modelleren. Feature Structures kunnen gebruikt worden om grammatica’s kort en leesbaar te houden. Ze geven een taalkundige verantwoording voor de argumenten van DCG’s. Discourse Representation Structures vormen een representatie die tussen natuurlijke taal en eerste-orde-logica ligt. Ze zijn even expressief als eerste-orde-logica maar een aantal concepten in de natuurlijke taal zijn beter te modelleren met een DRS.

Hoofdstuk 2

Eerder werk

In het verleden zijn er al meerdere CNL's gemaakt. Sommige zijn ontworpen om de leesbaarheid van de specificaties te verhogen en hebben geen formele semantiek. Daarom heeft Kuhn [13] een classificatieschema ontwerpen voor CNL's genaamd PENS: `Precision` (hoe ambigu/formeel is de taal), `Expressivity` (welke problemen kunnen we uitdrukken), `Naturalness` (hoe vlot leest de taal), `Simplicity` (hoe simpel is de taal). In dezelfde paper lijst Kuhn ook een heleboel CNL's op met hun geschiedenis en nut alsook hun classificatie volgens het PENS-schema.

Meer in het algemeen is het omvormen van teksten in natuurlijke taal tot formele modellen al gebeurd in verschillende domeinen: in vereistenanalyse, in het paradigma van business rules, binnen de computationele linguïstiek en ten slotte in het domein van de kennisrepresentatie.

Deze sectie geeft een kort overzicht van wat er al gebeurd is in al deze domeinen. Voor een completer overzicht van CNL's, verwijzen we naar Kuhn [13].

2.1 Vereistenanalyse

Circe De tool Circe [1] wordt gebruikt in vereistenanalyse. De gebruiker moet een vocabularium, een set van substitutieregels en een specificatie in natuurlijke aanleveren. De tool probeert dan steeds de beste substitutieregel te vinden om zo de tekst geleidelijk aan te transformeren naar een formeel model. Het grote voordeel van de methode is dat er regelsets bestaan voor meerdere soorten modellen: data flow modellen, entity-relationship modellen, ...

Verder is het in Circe mogelijk om in het vocabularium woorden te taggen. De regels kunnen hier dan gebruik van maken om te bepalen of ze van toepassing zijn op bepaalde zinnen. Op die manier introduceert Circe types in het vocabularium.

Een voorbeeld (uit [1]): *"bron/UIT STUURT data/INFO NAAR doel/IN"*. De woorden *"stuurt"* en *"naar"* liggen vast in de regelset. De andere 3 woorden komen uit het vocabularium. Deze moeten een bepaalde tag hebben om te matchen met de regel. Er zijn dus drie types in deze zin: entiteiten die informatie kunnen versturen, entiteiten die informatie kunnen ontvangen en informatie die verzonden kan worden.

Circe is niet echt een CNL. Zinnen die niet matchen met een substitutieregel worden ook niet omgezet in een formeel model. De specificatie kan dus zinnen met en zonder formele betekenis mengen. Er is geen sprake van constructieregels. Alleen (delen van) zinnen die matchen met een substitutieregel krijgen een formele betekenis.

2.2 Business rules

Binnen het paradigma van business rules spelen zowel natuurlijke talen als formele talen een grote rol. SBVR Structured English (SBVR-SE) [14] en RuleSpeak [19] zijn 2 CNL's die proberen om ambiguïteit in de natuurlijke taal te verminderen. Deze talen focussen echter vooral op het menselijke aspect [17]. Het doel van deze talen is om ambiguïteit uit de specificatie te halen en niet zozeer om automatisch natuurlijke taal om te zetten naar een formele voorstelling. RuleCNL [17] heeft wel dit doel.

RuleCNL splitst de specificatie op in twee delen: een vocabularium en de regels zelf. Het vocabularium bestaat uit substantieven en werkwoorden alsook hoe deze in verhouding staan tot elkaar. Bijvoorbeeld *"Auto heeft wiel"* geeft aan dat er een relatie kan bestaan tussen een auto en een wiel. Het vocabularium in RuleCNL is dus getypeerd. Voor de regels zelf bestaat er een contextvrije grammatica waaraan de zinnen moeten voldoen.

Om de gebruikers te helpen bij het schrijven van de zinnen in RuleCNL, is er een plug-in voor de Eclipse IDE die automatisch zinnen kan aanvullen en de structuur van bestaande zinnen aangeeft door het kleuren ervan. Bovendien is er een visuele representatie van het domein om de gebruiker te helpen bij het schrijven van het vocabularium.

2.3 Computationele linguïstiek

Er zijn reeds 2 belangrijke CNL's opgesteld die vertaald kunnen worden naar formele modellen: Attempto Controlled English (ACE) en Processable English (PENG). Beide talen lijken op elkaar en hebben gelijkaardige tools om mee te werken. Ze komen ook allebei uit de computationele linguïstiek en zijn ingebakken in taalkundige frameworks die gemaakt zijn om de semantiek van natuurlijke taal in het algemeen te vatten. In de papers over ACE en PENG wordt er niet zoveel gesproken over deze taalkundige aspecten omdat een achtergrond in de (computationele) linguïstiek verondersteld wordt.

Hierna volgt eerst een bespreking van ACE en dan van PENG. Bij de bespreking van PENG ligt de nadruk op de gelijkenissen en verschillen met ACE.

2.3.1 Attempto Controlled English (ACE)

Attempto Controlled English [11] is een gecontroleerde natuurlijke taal voor kennisrepresentatie. Het is een subset van Engels die naar een subset van eerste-orde-logica vertaalt. Het is een formele taal: elke zin in ACE heeft slechts één betekenis, ook al is de zin ambigu in het Engels. Om te bepalen welke van de betekenissen de *correcte* betekenis is, moet men de interpretatieregels volgen. Omdat dit soms nogal ingewikkeld is, kan men gebruik maken van de parafraseertool van ACE. Deze tool zet de interne representatie terug om naar één of meerdere zinnen in ACE. Op die manier kan men niet alleen de betekenis van de zin leren, maar ook de taal zelf. Deze parafrasering leunt echter zeer nauw aan bij de formele taal. Hierdoor is ze niet altijd toegankelijk voor iemand zonder een achtergrond in formele talen.

Bijvoorbeeld de zin *"Everybody is not present."* heeft 2 betekenissen in het Engels: *"Everybody is absent"* en *"Somebody is absent"*. In ACE is de eerste betekenis de *correcte*. Deze zin wordt geparafraseerd als *"If there is somebody X1 then it is false that X1 is present."*[1]. Dit is al een vrij moeilijke parafrasering om te begrijpen terwijl de originele zin nog redelijk simpel is.

[1] De parafrasering komt van de Attempto Parsing Engine (http://attempto.ifi.uzh.ch/ape/)

ACE is een general purpose CNL: Het bevat een ingebouwd vocabularium. De gebruiker moet dus zelf geen vocabularium opstellen en kan direct beginnen met het schrijven van de specificatie. Het nadeel aan deze aanpak is dat ACE dus ook geen domeinkennis kan gebruiken voor het analyseren van de zinnen. Sommige constructies moeten daarom met een koppelteken geschreven worden. Zo wordt er in [12] het voorbeeld gegeven van *"A student is interested-in a course"* en *"A student is interested in a classroom"*.

Op die manier probeert ACE sommige ambiguïteiten op te lossen. Een gelijkaardige truc wordt bijvoorbeeld ook gebruikt om de voorrang van *"en"* en *"of"* op te lossen. Standaard heeft *"en"* voorrang. Maar als de *"en"* voorafgegaan wordt door een komma, dan heeft *"of"* voorrang. [12] geeft het voorbeeld *"A client {enters a red card or enters a blue card}, and enters a code."*

In andere gevallen kiest ACE gewoon hoe de zin geïnterpreteerd moet worden op basis van een set van interpretatieregels. Zo slaat de *"manually"* in *"A customer who enters a card manually types a code."* [12] op *"enters"* en niet op *"types"* omdat een bijwoord bij voorkeur achter het werkwoord staat. (De parafrasering is in dit geval wel makkelijk te begrijpen maar vrij lang: *"There is a customer X1. The customer X1 types a code. The customer X1 enters a card manually."*). Merk ook op dat het onbepaalde lidwoord aanleiding geeft tot een existentiële quantor. In dit geval is de universele quantor echter beter geschikt.

Één van de sterke punten van ACE is haar coreferentie-analyse. Dit is het onderzoeken van welke woordgroepen naar hetzelfde concept verwijzen. Neem bijvoorbeeld de zinnen *"Een man heeft een vrouw. Hij is gelukkig."*. Hierin is *"Hij"* een anaforische referentie (een referentie naar een woordgroep die eerder komt) naar *"een man"*. ACE kan deze coreferenties correct analyseren. ACE doet dit door haar embedding in Discourse Representation Theory, een taalkundig framework. Men kan hier redelijk ver in gaan. Zo worden de zinnen *"There is a red house and there is a blue house. The red house is large."* correct geanalyseerd. Doordat zinnen in ACE als één geheel vertaald worden, kan men dus lange zinnen met veel bijzinnen herschrijven in meerdere kortere zinnen. Dit kan de leesbaarheid van een specificatie vergroten.

Origineel was ACE bedoeld voor het opstellen van specificaties voor software. Ondertussen kent de taal al meerdere toepassingen, in verschillende domeinen. Er zijn ook meerdere tools die overweg kunnen met ACE als input.

Zo is er de Attempto Parsing Engine APE die ACE zinnen omzet naar Discourse Representation Structures. APE geeft ook een parafrasering van de invoertekst. Zodat de gebruiker kan controleren of de tool de tekst op de juiste manier leest. Bovendien kan APE waarschuwingen geven bij mogelijke problemen. Bijvoorbeeld het gebruik van een anaforische referentie zonder een antecedent waarnaar deze anafoor kan verwijzen.

Verder is er de Attempto Reasoner RACE. Deze tool kan controleren of een specificatie consistent is. Indien niet, zal de tool zeggen welke zinnen met elkaar in conflict zijn. Op die manier weet de gebruiker dat er een fout is en waar deze zich ongeveer bevindt. Daarnaast kan de tool vragen in natuurlijke taal beantwoorden. RACE antwoordt niet alleen op de vraag maar geeft ook de zinnen die nodig zijn om te bewijzen dat het gegeven antwoord juist is. Ten slotte kan RACE bewijzen of een bepaalde zin het logische gevolg is van de specificatie. Op die manier kan de gebruiker testen of de specificatie correct is.

APE en RACE zijn de twee belangrijkste tools. Er zijn er echter nog veel meer. Zo is er de ACE View Protégé plug-in. Dit is een plug-in die de vertaling tussen Web

Ontology Language (OWL) en ACE voorziet binnen de Protégé-omgeving (een editor voor het maken van ontologiën). Op die manier ziet de gebruiker enkel ACE zinnen en hoeft dus de formele OWL taal niet te kennen om met bestaande modellen om te gaan of om nieuwe modellen te maken. Ten slotte is er AceRules. Hiermee kan de gebruiker de zinnen die geïmpliceerd worden door de specificatie te weten komen.

Over het algemeen is ACE een zeer uitgebreide taal. Veel Engelse zinnen zijn geldige ACE zinnen, echter niet allemaal. Hierdoor is het moeilijk om de taal te leren. Het is immers niet super duidelijk wat toegestaan is en wat niet. Volgens Fuchs et al. [11] heeft een gebruiker 2 dagen nodig om de taal te leren.

2.3.2 Processable English (PENG)

Andere talen zoals PENG [21] zijn makkelijker om te leren dan ACE. PENG is net als ACE een CNL die vertaalt naar Discourse Representation Structures. In tegenstelling tot ACE bevat PENG geen groot ingebouwd lexicon. De gebruiker moet zelf de woorden aanbrengen die gebruikt worden. De gebruiker kan dit doen tijdens het bewerken en moet dus niet op voorhand aangeven wat het lexicon is. De categorieën voor deze domeinspecifieke woorden zijn substantief, adjectief, werkwoord of bijwoord. PENG biedt ook de mogelijkheid om synoniemen of afkortingen te introduceren. Op die manier kan de specificatie vlotter gemaakt worden. Men kan echter geen types meegeven in het lexicon. Men kan dus niet zeggen dat het werkwoord *"ademen"* enkel uitgevoerd kan worden doormensen of dieren en niet door objecten. Naast de domeinspecifieke woorden kent PENG een aantal functionele woorden die ingebakken zitten in de taal, zoals lidwoorden en voegwoorden (*"de man die"*). Deze woorden helpen PENG om de zinsconstructies te herkennen.

Net zoals ACE kent PENG het principe van constructieregels en interpretatieregels. De constructieregels bepalen welke zinnen deel zijn van de taal. Bij PENG zijn deze regels eenvoudiger omdat de taal zeer simpel gehouden is. Hierdoor is het makkelijker om zinnen te maken in de taal. Over het algemeen lijken de constructieregels van PENG en ACE veel op elkaar.

De interpretatieregels bepalen hoe de zin vertaald wordt naar logica. Zo is er een interpretatieregel voor hoe anaforische referenties opgelost worden. Deze regel is gelijkaardig in ACE en PENG. Andere regels bepalen welke functiewoorden sterker binden. Zowel ACE als PENG gebruiken hiervoor dezelfde volgorde als in eerste-orde-logica. ACE staat wel uitzonderingen toe door het toevoegen van komma's. PENG houdt de taal simpel en staat dit niet toe.

PENG is makkelijker om te leren dan ACE omwille van ECOLE [23]. Een tool die suggesties geeft over de woordcategorieën die kunnen volgen op een bepaalde zin. Zo geeft Schwitter [23] het voorbeeld van *"Een"* dat gevolgd kan worden door een `adjectief` of een `substantief`. Indien de gebruiker de woordcategorie niet kent, kan hij doorklikken op die categorie voor een aantal concrete mogelijkheden. Op die manier moet de gebruiker enkel de woordcategorieën leren en niet de geldige zinsconstructies.

Naast de suggestietool bestaat er voor PENG, net zoals voor ACE, ook een parafraseertool. Deze tool herschrijft de invoer zodat het duidelijker is hoe PENG de zin begrepen heeft. Anaforische referenties worden bijvoorbeeld omgezet in de naamwoordgroep waarnaar ze verwijzen.

Net zoals voor ACE zinnen is het ook voor een PENG zinnen mogelijk om te controleren of ze een consistente specificatie vormen [22]. Daarnaast is het mogelijk om te controleren op redundantie. Indien een zin al door een andere zin geïmpliceerd wordt,

Woord	Categorie	λ-ASP-expressie
Birds	np	$\lambda x.bird(x)$
Fly	s\np	$\lambda P.fly(X) \leftarrow P(X)$

Tabel 2.1: Een lexicon voor de woorden birds en fly in een CCG grammatica (uit [3])

hoeft deze niet expliciet deel uit te maken van de specificatie. Op die manier kan de specificatie kort gehouden worden, wat de leesbaarheid verhoogd.

Naast de general purpose CNL bevat PENG ook een subset specifiek voor het semantische web: PENG-D [24]. Deze subset kan vertaald worden naar description logic (OWL DL), een subset van eerste-orde-logica. PENG-D kan dus gezien worden als het alternatief voor de ACE View Protégé plug-in. Schwitter [25] vermeldt drie klassieke manieren van voorstellen van een ontologie (N-Triples, RDF/XML en OWL Abstract) en toont dan verder aan dat PENG een vierde manier is om hetzelfde voor te stellen. De paper toont dit aan door verschillende constructies uit OWL te mappen op zinnen in PENG. Het grote voordeel van PENG t.o.v. de andere voorstellingswijzen is dat PENG ook leesbaar en begrijpbaar is voor de mens.

2.4 Kennisrepresentatie

In het domein van kennisrepresentatie hebben Baral et al. [3] natuurlijke taal reeds vertaald naar Answer Set Programs (ASP). Ze maken hiervoor gebruik van Combinatorische Categorische Grammatica (CCG) en λ-calculus. Een CCG bestaat uit een aantal basiscategorieën zoals `s` (zin) en `np` (naamwoordgroep) en afgeleide categorieën zoals `s\np`[2] en `(s\np)/np`[3] [3]. Zo wordt een onovergankelijk werkwoord voorgesteld als een `s\np`. Verder bevat een CCG een aantal regels die bepalen wat de categorie is van een combintatie van woordgroepen. Zo is er een regel die zegt dat $\alpha\beta$ van categorie `B` is als α categorie `A` heeft en β categorie `B\A` [3]. Dankzij deze regel kunnen we afleiden dat als we een onovergankelijk werkwoord (`s\np`) langs links combineren met een naamwoordgroep (`np`), we dan een zin (`s`) krijgen. Op deze manier kunnen we een parse tree opstellen waarbij we telkens 2 woordgroepen combineren totdat de zin uiteindelijk categorie `s` krijgt.

Naast een categorie heeft elk woord ook een betekenis uitgedrukt in een uitbreiding op de λ-calculus. In deze uitbreiding kunnen ook ASP-expressies voorkomen. De betekenis van een woordgroep is de combinatie van de betekenis van de woorden. Hierbij gebruiken we de CCG parse tree om de volgorde te bepalen. We verduidelijken met een voorbeeld (grotendeels overgenomen uit Baral et al. [3]). We beschouwen het lexicon zoals weergegeven in tabel 2.1 voor de zin *"Birds fly"*. De combinatie van de woorden *birds* en *fly*, is van de categorie `s` zoals hierboven reeds uitgelegd. De overeenkomstige lamda-expressies moeten nu op een gelijkaardige manier gecombineerd worden: $\lambda_{birds\ fly} = \lambda_{fly}(\lambda_{birds})$.

$$\begin{aligned} \lambda_{birds\ fly} &= \lambda_{fly}(\lambda_{birds}) \\ &= (\lambda P.fly(X) \leftarrow P(X))(\lambda x.bird(x)) \\ &= fly(X) \leftarrow (\lambda x.bird(x))(X) \\ &= fly(X) \leftarrow bird(X) \end{aligned} \tag{2.1}$$

[2] een `S\NP` is een woordgroep die indien gecombineerd met een `np` langs links een zin vormt

[3] een `(S\NP)/NP` is een woordgroep die indien met een `np` gecombineerd langs rechts en langs links een zin vormt

Elk woord heeft een betekenis die men kan zien als een ASP-expressie met gaten erin die opgevuld worden door de combinatie met andere woorden. De manier van combineren van de λ-expressies hangt af van de categorieën van de woorden. Het nadeel aan deze aanpak is dat elk woord (minstens) één betekenis moeten hebben in het formaat van een λ-ASP-expressie. Constantini et al. [9] lossen dit probleem deels op door gebruik te maken van λ-ASP-expressie-templates. Sommige woorden hebben nog steeds een eigen λ-ASP-expressie, voor de andere kan er één afgeleid worden uit een λ-ASP-expressie-template. Zo is $\lambda x.\langle noun\rangle(x)$ de template voor substantieven. Het gedeelte $\langle noun\rangle$ moet vervagen worden door een specifieke instantie. Het woord *birds* heeft zo nog steeds dezelfde betekenis.

Zo'n templates werken echter niet voor alle woorden. Daarom hebben Baral et al. [4] een methode bedacht om van de betekenis van een zin en de betekenis van een aantal woorden, de betekenis van andere woorden af te leiden. Ze doen dit niet langer met λ-ASP-expressies maar vervangen ASP door eerste-orde-logica: λ-FOL-expressies. De grammatica en manier van combineren blijft echter hetzelfde.

Deze techniek gebruiken Baral et al. [2] om automatisch een grammatica en semantiek te leren voor logigrammen op basis van andere logigrammen en hun vertaling in een ASP programma. Hiervoor gebruiken ze één ASP-ontologie die toepasbaar is voor vele logigrammen. Ze bewerken hierbij de natuurlijke taal van de logische puzzels een beetje om anaforische referenties te verwijderen. De auteurs benadrukken echter dat dit geen CNL is omdat de grammatica op voorhand niet gedefinieerd is maar geleerd wordt uit de gegeven logische puzzels. Ze maken hiervoor gebruik van een probabilistische CCG. Tot 83% van de puzzels kunnen ze correct oplossen. De andere puzzels falen o.a. omdat bepaalde zinsconstructies niet voorkwamen in de trainingsdata. Belangrijk om op te merken is dat vele woorden (zoals *about*, *on*, *the*) geen betekenis krijgen omdat ze in de logische puzzel geen rol spelen. Er is dus sprake van overfitting op het domein van de logische puzzels.

Verder kan deze vertaling enkel gebruikt worden om logigrammen op te lossen. Hun systeem ondersteunt geen andere soorten inferenties op die logigrammen.

Hoofdstuk 3

Een overzicht van het systeem

In dit hoofdstuk beschrijven we het systeem achter deze thesis. Dit systeem stelt een formele specificatie op van een logigram.

Eerst leggen we uit wat het systeem met zo'n specificatie kan doen. Daarna geven we de stappen die het systeem onderneemt om de specificatie op te stellen. Ten slotte geven we een voorbeeld van de in- en uitvoer van het systeem.

3.1 Inferenties op logigrammen

In deze thesis stellen we een gecontroleerde natuurlijke taal op. Deze CNL wordt afgeleid uit de zinnen van de eerste tien logigrammen uit Puzzle Baron's Logic Puzzles Volume 3 [20]. We geven deze taal een formele semantiek. Daardoor kan men deze taal zien als een kennisrepresentatietaal (voor logigrammen).

Een logigram dat uitgedrukt is in deze CNL kan dan omgezet worden tot een formele specificatie die als invoer van een *Knowlegde Base System* kan dienen. Het KBS kan dan nadenken over het logigram. Zo een systeem kan

- Een logigram automatisch oplossen
- Gegeven een (partiële) oplossing, aangeven aan welke zinnen voldaan is, welke zinnen nog nieuwe informatie bevatten en welke zinnen niet overeenkomen met de gegeven oplossing.
- Gegeven een partiële oplossing, automatisch een subset van de zinnen afleiden die gebruikt kan worden om de oplossing verder aan te vullen. De gebruiker kan zo een hint krijgen om het logigram verder op te lossen.
- Gegeven een set van zinnen, aangegeven welke oplossingen nog mogelijk zijn. Dit kan de auteur helpen bij het schrijven van nieuwe logigrammen. Afhankelijk van welke oplossingen nog mogelijk zijn, kan de auteur een nieuwe zin toevoegen om een aantal van de mogelijkheden te elimineren tot er maar één oplossing overblijft.

3.2 De stappen

Om een specificatie op te stellen van een logigram vanuit de zinnen in een gecontroleerde natuurlijke taal en een logigram-specifiek lexicon, neemt het systeem de volgende stappen:

1. Het vertaalt de zinnen naar logica met het (aangepaste) framework van Blackburn en Bos. Hierbij verzamelt het informatie i.v.m. de types van woorden.

2. Het systeem gebruikt die informatie i.v.m. de types van woorden om de domeinen van een logigram af te leiden. Indien nodig stelt het systeem vragen aan de gebruiker om te helpen bij deze type-inferentie.
3. Gebaseerd op deze domeinen en taalkundige informatie stelt het systeem een formeel vocabularium op
4. Het systeem formuleert ook een aantal axioma's die de impliciete assumpties van een logigram modelleren
5. Ten slotte geeft het deze specificatie (bestaande uit het formele vocabularium en de theorie met de axioma's en de vertaling van de zinnen van het logigram) aan IDP [10]. Het systeem kan dan aan IDP vragen om de oplossing te berekenen (of een andere soort van inferentie toe te passen op de specificatie).

Voor de eerste stap maken we gebruik van het framework van Blackburn en Bos (hoofdstuk 4). We stellen een set van lexicale categorieën (hoofdstuk 5) en een grammatica (hoofdstuk 6) op om te gebruiken in dit framework. We breiden het framework ook uit met types (hoofdstuk 7). Daardoor kunnen we vertalen naar een getypeerde logica en kunnen we de informatie i.v.m. de types van woorden verzamelen.

Voor de tweede stap veronderstellen we dat elk woord exact één type heeft per logigram (zie ook hoofdstuk 7). Indien deze assumptie niet voldoende informatie verschaft, stelt het systeem een aantal vragen aan de gebruiker totdat het de domeinen heeft afgeleid.

Met behulp van de afgeleide domeinen en taalkundige informatie kan het systeem de specificatie vervolledigen met een formeel vocabularium en de impliciete axioma's (hoofdstuk 8).

In hoofdstuk 9 testen we dit systeem uit op ongezien logigrammen. Hierbij controleren we of het systeem de correcte oplossing berekent van de logigrammen.

3.3 Een voorbeeld specificatie

In appendix A staat de specificatie die het systeem genereert voor logigram 1 uit Puzzle Baron's Logic Puzzles Volume 3 [20]. Deze appendix bevat ook de invoer die het systeem nodig heeft en alle vragen die het systeem stelt aan de gebruiker voor deze logigram.

Hoofdstuk 4

Een framework voor semantische analyse

In dit hoofdstuk bespreken we het framework van Blackburn en Bos [5, 6] voor semantische analyse. Het framework bestaat uit een lexicon (het vocabularium), de grammatica, de semantiek van de grammaticale regels en de semantiek van de woorden in het lexicon. Het hele framework is gebaseerd op lambda-calculus en Frege's compositionaliteitsprincipe dat stelt dat de betekenis van een woordgroep enkel afhangt van de betekenissen van de woorden waaruit ze bestaat.

Dit hele hoofdstuk is een samenvatting van de relevante hoofdstukken van de boeken van Blackburn en Bos [5, 6]. De formules komen grotendeels overeen met die uit hun boeken en uit de code die erbij hoort. De aanpassingen die zijn gebeurd, dienen vooral om de leesbaarheid te verhogen. De nadruk op de signaturen van de verschillende categorieën is nieuw, al komt de getypeerde lambda-calculus met de twee types wel aan bod in appendix B van hun eerste boek [5].

4.1 Lexicon

Het lexicon bestaat uit een opsomming van alle woorden met een aantal (taalkundige) features (zoals de categorie van het woord). Tabel 4.1 geeft een voorbeeld van een lexicon. Het lexicon is meer dan een woordenboek. Het bevat alle woordvormen, niet enkel de basisvorm. Zo komt "love" drie keer voor in het lexicon. "love" zelf komt één keer voor als infinitief en één keer als meervoud van de tegenwoordige tijd. "loves" is dan weer het enkelvoud van de tegenwoordige tijd.

De meeste woordvormen hebben ook een feature `Symbool` die zal gebruikt worden bij de semantiek van de woorden. Deze feature maakt het onderscheid tussen de verschillende woorden van dezelfde categorie. Het symbool komt overeen met de naam van de constante of van het predicaat die het woord introduceert in het formeel vocabularium (zie sectie 8.1).

4.2 Grammatica

De grammatica bepaalt welke woorden samen woordgroepen vormen, welke woordgroepen samen andere woordgroepen vormen en welke woordgroepen een zin vormen. Op die manier ontstaat er een boom van woorden.

Woordvorm	Categorie	Symbool	Andere features
man	zelfstandig naamwoord	man	num=sg
men	zelfstandig naamwoord	man	num=pl
woman	zelfstandig naamwoord	woman	num=sg
women	zelfstandig naamwoord	woman	num=pl
John	eigennaam	John	
sleep	onovergankelijk werkwoord	sleeps	inf=inf
sleeps	onovergankelijk werkwoord	sleeps	inf=fin, num=sg
sleep	onovergankelijk werkwoord	sleeps	inf=fin, num=pl
love	overgankelijk werkwoord	loves	inf=inf
loves	overgankelijk werkwoord	loves	inf=fin, num=sg
love	overgankelijk werkwoord	loves	inf=fin, num=pl
a	determinator	/	type=existential
every	determinator	/	type=universal

Tabel 4.1: Een voorbeeld van een lexicon

Een simpele grammatica Grammatica 4.1 bevat een simpele grammatica. Een zin bestaat in deze grammatica uit een `np` gevolgd door een `vp`, beiden met hetzelfde getal. Een naamwoordgroep (noun phrase of `np`) is een woordgroep die naar één of meerdere entiteiten verwijst. Zo'n groep wordt ook wel een nominale constituent genoemd. Een verb phrase (`vp`) of verbale constituent drukt meestal een actie uit.

```
1 s() --> np([num:Num]), vp([num:Num]).
2 s() --> [if], s(), s().
3 np([num:sg]) --> pn().
4 np([num:Num]) --> det([num:Num]), n([num:Num]).
5 vp([num:Num]) --> iv([num:Num]).
6 vp([num:Num]) --> tv([num:Num]), np([num:_]).
```

Grammatica 4.1: Een simpele grammatica

Een zin kan ook bestaan uit het functiewoord "if", gevolgd door twee zinnen (bijvoorbeeld "If a man breathes, he lives"). Functiewoorden zijn deel van de grammatica en komen niet voor in het lexicon. Ze helpen om de structuur van de zin te herkennen. De betekenis van deze woorden komt via de semantiek van de grammatica naar boven.

Een naamwoordgroep (`np`) kan bestaan uit een eigen naam (proper name of `pn`) of uit een determinator (`det`, bijvoorbeeld een lidwoord) en een zelfstandig naamwoord (noun of `n`) die overeenkomen in getal. Een eigennaam is in deze grammatica altijd in het enkelvoud.

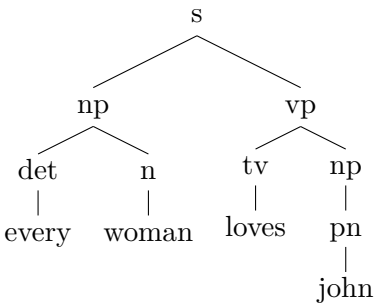
Een verbale constituent (`vp`) bestaat uit een onovergankelijk werkwoord (intransitive verb of `iv`) of uit een vergankelijk werkwoord (transitive verb of `tv`) gevolgd door een nieuwe naamwoordgroep (als lijdend voorwerp). Het werkwoord moet in getal overeenkomen met het getal van de verbale constituent. Daardoor zal het getal van het werkwoord en het onderwerp altijd overeenkomen.

Grammatica 4.1 bevat een aantal categorieën (`pn`, `det`, `n`, `iv` en `tv`) die overeenkomen met de lexicale categorieën. Voor deze categorieën wordt er in het lexicon gezocht naar een woord met de juiste features.

Bovenstaande grammatica is nog heel beperkt. De moeilijkheid ligt erin om de grammatica simpel te houden maar toch zoveel mogelijk gewenste zinnen toe te laten.

Om logigrammen automatisch te kunnen vertalen moet er dus een grammatica opgesteld worden die de zinnen van deze logigrammen omvat.

Een boom Op basis van deze grammatica kunnen we ook een parse tree opbouwen voor elke geldige zin. Zo wordt "Every woman loves john omgezet in de boom



Op elke knoop in deze boom zullen we later Frege's compositionaliteitsprincipe toepassen: de betekenis van een woordgroep is gelijk aan een combinatie van de betekenissen van de woord(groep)en waaruit ze bestaat.

Conclusie De grammatica bepaalt welke combinaties van woorden zinnen vormen. Ze bepaalt dus welke zinnen in de taal liggen en welke er buiten vallen. Bovendien geeft de grammatica ons een boom. Deze boom zullen we gebruiken om de betekenis van de woorden naar boven toe te laten propageren volgens Frege's compositionaliteitsprincipe tot de betekenis van de zin.

4.3 Semantiek

Het lexicon bepaalt welke woorden gebruikt mogen worden, de grammatica hoe deze woorden een zin kunnen vormen. De vraag die nog rest is welke betekenis de zin heeft. Daarvoor doen we een beroep op de lambda-calculus. Eerst bespreken we hoe we de betekenis van een woordgroep kunnen afleiden uit de betekenis van de woordgroepen waaruit ze bestaan. Daarna bespreken we de betekenis van de woorden uit het lexicon.

4.3.1 Semantiek van de grammaticale regels

Een getypeerde lambda-calculus Voor de betekenis van de taal zullen we gebruiken maken van een getypeerde lambda-calculus. Er zijn twee basistypes in onze lambda-calculus, namelijk e voor entiteiten en t voor waarheidswaarden (*waar* en *onwaar*). Ten slotte is er de type-constructor voor een functie-type die we noteren met $\rightarrow$. Zo is $\lambda x \cdot man(x)$ een lambda-expressie van type $e \rightarrow t$. Elk woord zal een lambda-formule als betekenis krijgen. Deze formules zullen gecombineerd worden tot één formule voor de zin die geen lambda's meer bevat. Die formule vormt de betekenis van de zin. De betekenis van een zin is dus van type t.

Frege's compositionaliteitsprincipe Voor de betekenis van de grammatica berusten we op Frege's compositionaliteitsprincipe: De betekenis van een woordgroep bestaat uit een combinatie van de betekenissen van de woorden of woordgroepen waaruit ze bestaat. Deze combinatie kan afhangen van de manier waarop de woorden gecombineerd worden. Op deze manier propageren we de semantiek van de woorden (die de bladeren in de boom vormen) naar boven toe om zo de betekenis van de zin te verkrijgen. We

hernemen bovenstaande grammaticale regels nu en voegen de semantiek toe. Later zullen we aantonen dat deze simpele combinatieregels tot een zinnig resultaat kunnen leiden.

Grammaticale regel	**Semantiek**
`s --> np([num:Num]), vp([num:Num]).`	$[\![s]\!] = [\![vp]\!]\,([\![np]\!])$
`s --> [if], s1, s2.`	$[\![s]\!] = \dfrac{\quad}{[\![s_1]\!] \Rightarrow [\![s_2]\!]}$
`np([num:sg]) --> pn.`	$[\![np]\!] = [\![pn]\!]$
`np([num:Num]) --> det([num:Num]), n([num:Num]).`	$[\![np]\!] = [\![det]\!]\,([\![n]\!])$
`vp([num:Num]) --> iv([num:Num]).`	$[\![vp]\!] = [\![iv]\!]$
`vp([num:Num]) --> tv([num:Num]), np([num:_]).`	$[\![vp]\!] = [\![tv]\!]\,([\![np]\!])$

Tabel 4.2: De semantiek van grammatica 4.1

Tabel 4.2 geeft een overzicht van de semantiek van grammatica 4.1. Een idee achter het framework is om de betekenis van de zinnen zoveel mogelijk in de betekenis van de woorden te stoppen. De betekenis van een grammaticale regel wordt, indien mogelijk, beperkt tot lambda-applicaties. Voor woordgroepen die maar uit één woord bestaan is de betekenis van de woordgroep gelijk aan die van het woord zelf. Voor de meeste andere woordgroepen is de betekenis enkel een lambda-applicatie van de betekenissen van de woordgroepen waaruit ze bestaan. Meestal heeft een woordgroep één woord dat essentieel is aan de groep. De woordgroep is hier dan ook naar vernoemd[1]. Dit woord is de functor van de lambda-applicatie, het andere woord het argument. Enkel voor de speciale zinsstructuur van de conditionele zin is er ook een speciale constructie nodig in de semantiek. Indien we het functiewoord "if" naar het lexicon zouden verhuizen, dan zouden twee lambda-applicaties volstaan[2].

4.3.2 Semantiek van het lexicon

We weten nu hoe we de semantiek van de woorden kunnen combineren tot de semantiek van de woordgroepen en bij uitbreiding tot die van een zin. Er ontbreekt enkel nog de semantiek van de woorden zelf.

Voor we de betekenis van woorden kunnen opstellen moeten we eerst de signatuur achterhalen. Cruciaal voor het framework is dat elke grammaticale categorie exact 1 signatuur heeft. Zodanig dat we altijd woordgroepen van dezelfde categorie kunnen uitwisselen. We gebruiken de functie τ om de signatuur aan te duiden.

We beginnen met de signatuur van een zelfstandig naamwoord (`n`) te achterhalen. Daarna bekijken we achtereenvolgens de nominale constituent (`np`), de eigennaam (`pn`) en het lidwoord (`det`). Ten slotte bekijken we de hiërarchie van de verbale constituenten: de verbale constituent zelf (`vp`), onovergankelijk werkwoord (`iv`) en overgankelijk werkwoord (`tv`).

De signatuur van een zelfstandig naamwoord. Een zelfstandig naamwoord, zoals bijvoorbeeld *man*, stelt een verzameling entiteiten voor. In het voorbeeld van *man* stelt dit de verzameling van alle mannen voor. In logica wordt zo'n verzameling typisch voorgesteld door een unair predicaat. In deze setting gebeurt net hetzelfde: we stellen

[1] Voor de semantiek van een naamwoordgroep wordt het lidwoord toch als het belangrijkste woord aanzien door Blackburn en Bos

[2] $[\![if]\!] = \lambda S1 \cdot \lambda S2 \cdot \dfrac{\quad}{S1 \Rightarrow S2}$ en $[\![s]\!] = [\![if]\!]\,([\![s_1]\!])\,([\![s_2]\!])$

een zelfstandig naamwoord voor als een functie van entiteiten naar waarheidswaarden, of in formule $e \to t$. Dit is de functionele variant van een unair predicaat. De functie van een substantief bepaalt of een bepaalde entiteit omschreven kan worden met dat substantief of niet.

De signatuur voor een nominale constituent (np) Een nominale constituent of naamwoordgroep is een woordgroep die kan dienen als onderwerp of lijdend voorwerp. De betekenis ervan is een verwijzing naar een entiteit (of een groep van entiteiten). In een zin zeggen we altijd iets over deze entiteiten, bijvoorbeeld "a man sleeps". Een zin is waar als de entiteit(en) van de naamwoordgroep voldoen aan een bepaalde eigenschap, bijvoorbeeld als er een man is die de eigenschap heeft dat hij slaapt. Dat is wat signatuur

$$\tau(np) = (e \to t) \to t$$

uitdrukt. Het eerste argument van type $e \to t$ is een eigenschap waaraan entiteiten kunnen voldoen. Bijvoorbeeld voor "sleeps" is dit (in eerste-orde-logica) $\lambda x \cdot sleeps(x)$.

Een naamwoordgroep voldoet aan een eigenschap als er *genoeg* entiteiten zijn die omschreven worden door de naamwoordgroep en die voldoen aan de eigenschap. Een voorbeeldimplementatie voor "a man" (in eerste-orde-logica) is

$$\lambda E \cdot \exists x \cdot man(x) \wedge E(x)$$

Er moet namelijk minstens één man zijn die aan de eigenschap E voldoet. In het geval van "every man" moeten alle mannen voldoen aan de eigenschap E

$$\lambda E \cdot \forall x \cdot man(x) \Rightarrow E(x)$$

Eigennaam en lidwoord De signatuur van een eigennaam is gelijk aan die van een naamwoordgroep. Dat volgt uit de semantiek van de grammatica.

$$\tau(pn) = \tau(np) = (e \to t) \to t$$

Uit de semantiek van de grammatica volgt ook dat de signatuur van een lidwoord gelijk is aan die van een zelfstandig naamwoord naar een naamwoordgroep.

$$\tau(det) = \tau(n) \to \tau(np) = (e \to t) \to (e \to t) \to t$$

Bijvoorbeeld voor "every" wordt dit (met R een restrictie vanuit het substantief)

$$\lambda R \cdot \lambda E \cdot \forall x \cdot R(x) \Rightarrow E(x)$$

Verbale constituent De signatuur voor een verbale constituent (`vp`) volgt opnieuw uit de grammatica en de andere signaturen

$$\tau(vp) = \tau(np) \to \tau(s) = ((e \to t) \to t) \to t$$

Een zin is waar als het onderwerp voldoet aan de eigenschap die het werkwoord uitdrukt.

De signatuur van een onovergankelijk werkwoord (`iv`) is gelijk aan die van een verbale constituent (volgens de grammatica).

$$\tau(iv) = \tau(vp) = ((e \to t) \to t) \to t$$

De signatuur van een overgankelijk werkwoord (`tv`) wordt dan

$$\tau(tv) = \tau(np) \to \tau(vp) = ((e \to t) \to t) \to ((e \to t) \to t) \to t$$

Tabel 4.3 vat alle signaturen nog eens samen. Op basis van deze signaturen kunnen we de betekenis van het lexicon opstellen. Blackburn en Bos gebruiken hiervoor *semantische macro's*. Dat wil zeggen dat ze voor elke lexicale categorie een functie hebben die een woordvorm uit het lexicon afbeeldt op een lambda-expressie. Hiervoor worden enkel de lexicale features van de woordvorm in kwestie gebruikt.

Grammaticale categorie	**Signatuur**
Zin (`s`)	t
Naamwoordgroep (`np`)	$(e \to t) \to t$
Eigennaam (`pn`)	$(e \to t) \to t$
Zelfstandig naamwoord (`n`)	$(e \to t)$
Lidwoord (`det`)	$(e \to t) \to (e \to t) \to t$
Verbale constituent (`vp`)	$((e \to t) \to t) \to t$
Onovergankelijk werkwoord (`iv`)	$((e \to t) \to t) \to t$
Vergankelijke werkwoord (`tv`)	$((e \to t) \to t) \to ((e \to t) \to t) \to t$

Tabel 4.3: De signaturen van de grammaticale categorieën uit grammatica 4.1

Grammatica 4.1 telt 5 lexicale categorieën: `pn`, `n`, `det`, `iv` en `tv`.

- Een eigennaam (`pn`) introduceert een constante met als naam het symbool dat bij die eigennaam hoort. Een eigennaam voldoet aan de eigenschap E als de entiteit waarnaar verwezen wordt eraan voldoet. Meer formeel wilt dit zeggen dat de functiewaarde van die constante voor de functie E gelijk is aan de waarheidswaarde *waar* of $\llbracket pn \rrbracket = \lambda E \cdot E\,(Symbool)$ of voor "John" $\llbracket John \rrbracket = \lambda E \cdot E\,(John)$.
- Een zelfstandig naamwoord (`n`) test of een referentie kan omschreven worden met dat naamwoord of niet. $\llbracket n \rrbracket = \lambda x \cdot \begin{array}{|c|}\hline \\ \hline Symbool(x) \\ \hline \end{array}$. Bijvoorbeeld voor "man" $\llbracket man \rrbracket = \lambda x \cdot \begin{array}{|c|}\hline \\ \hline man(x) \\ \hline \end{array}$
- Een lidwoord of meer algemeen een determinator (`det`) introduceert een nieuwe referentie die een bepaalde scope heeft. Hier zijn er meerdere mogelijke vertalingen, één voor elk type van determinator. Een determinator heeft 2 argumenten: de *restriction* en de *nuclear scope*. De *restriction* R wordt opgevuld door het zelfstandig naamwoord (met eventuele bijzinnen). De *nuclear scope* S wordt opgevuld door de verbale constituent. Het is de eigenschap waaraan de naamwoordgroep moet voldoen.
  - Voor een universele determinator (zoals "every") moet de variabele met een universele kwantor gebonden zijn. In eerste-orde-logica krijgen we dan $\llbracket det_{universeel} \rrbracket = \lambda R \cdot \lambda S \cdot \forall x \cdot (R\,(x) \Rightarrow S\,(x))$. In DRS wordt dit

$$\llbracket det_{universeel} \rrbracket = \lambda R \cdot \lambda S \cdot \begin{array}{|c|}\hline \\ \hline \left\{ \begin{array}{|c|}\hline x \\ \hline \\ \hline \end{array} \oplus R\,(x) \right\} \Rightarrow S\,(x) \\ \hline \end{array}$$

  - De existentiële determinator (zoals "a") introduceert een variabele die gebonden is door een existentiële kwantor. De $\oplus$-operator vormt de logische

conjunctie tussen twee DRS-structuren.

$$[\![det_{existentieel}]\!] = \lambda R \cdot \lambda S \cdot \left\{ \begin{array}{|c|}\hline x \\ \hline \end{array} \oplus R(x) \oplus S(x) \right\}$$

– De negatieve determinator (zoals “no”): Deze determinator drukt uit dat er geen entiteit bestaat die tegelijk aan de restrictie R en de eigenschap S voldoet. Deze introduceert dus een negatie van een existentiële kwantor.

$$[\![det_{negatief}]\!] = \lambda R \cdot \lambda S \cdot \begin{array}{|c|}\hline \\ \hline \neg \left[\begin{array}{|c|}\hline x \\ \hline \end{array} \oplus R(x) \oplus S(x) \right] \\ \hline \end{array}$$

- Een onovergankelijk werkwoord (`iv`) neemt de naamwoordgroep die het onderwerp vormt als argument. Het moet testen of dit onderwerp O voldoet aan de eigenschap van het werkwoord $\lambda x \cdot \begin{array}{|c|}\hline \\ \hline Symbool(x) \\ \hline \end{array}$, namelijk of het onderwerp de *actie* van het werkwoord uitvoert.

$$[\![iv]\!] = \lambda O \cdot O \left(\lambda x \cdot \begin{array}{|c|}\hline \\ \hline Symbool(x) \\ \hline \end{array} \right)$$

- Een overgankelijk werkwoord (`tv`) is gelijkaardig maar krijgt twee naamwoordgroepen als argument. Het eerste argument is het lijdend voorwerp L, het tweede het onderwerp O. Er zijn meerdere vertaling mogelijk. Neem bijvoorbeeld de zin “Every man loves a woman”. Is er één vrouw waarvan alle mannen houden of kan dit een verschillende vrouw zijn voor elke man? Dit wordt een *Quantifier Scope Ambiguïteit* genoemd omdat de ambiguïteit ligt in de volgorde van de kwantoren. Zo worden de twee lezingen respectievelijk $\exists w \cdot woman(w) \wedge (\forall m \cdot man(m) \Rightarrow loves(m, w))$ en $\forall m \cdot man(m) \Rightarrow (\exists w \cdot woman(w) \wedge loves(m, w))$. Blackburn en Bos lossen deze ambiguïteiten op door de kwantoren in de vertaling in dezelfde volgorde te laten als in de natuurlijke taal[3]. Voor het overgankelijke werkwoord wordt dit

$$[\![tv]\!] = \lambda L \cdot \lambda O \cdot O \left(\lambda x_o \cdot L \left(\lambda x_l \cdot \begin{array}{|c|}\hline \\ \hline Symbool(x_o, x_l) \\ \hline \end{array} \right) \right)$$

Een entiteit x_o omschreven door het onderwerp voldoet aan de verbale constituent als voor die x_o het lijdend voorwerp voldoet aan de eigenschap $\lambda x_l \cdot \begin{array}{|c|}\hline \\ \hline Symbool(x_o, x_l) \\ \hline \end{array}$. We passen dit toe op “Every man loves a woman”. Een man x_o voldoet aan de verbale constituent (“loves a woman”) als er voor die man x_o een vrouw x_l is zodat $\begin{array}{|c|}\hline \\ \hline loves(x_o, x_l) \\ \hline \end{array}$ waar is. Het onderwerp (als geheel) voldoet aan de verbale constituent als elke man x_o voldoet aan bovenstaande eigenschap. Als er dus voor elke man x_o een vrouw x_l bestaat waarvan hij houdt. Voor elke man kan er dit een andere vrouw x_l zijn.

Merk op dat al deze lambda-expressies voldoen aan de signaturen van tabel 4.3

[3] Ze leggen daarnaast ook uit hoe men de andere lezingen kan verkrijgen. We verwijzen nar hoofdstuk 3 uit hun eerste boek [5] voor de details.

4.4 Een voorbeeld

In appendix B illustreren we het framework en bovenstaande formules aan de hand van de zin “If every man sleeps, a woman loves John”. De vertaling wordt

$$[\![s]\!] = \boxed{\begin{array}{l} \\ \boxed{\begin{array}{l} \\ \boxed{\begin{array}{l} x \\ man(x) \end{array}} \Rightarrow \boxed{\begin{array}{l} \\ sleeps(x) \end{array}} \end{array}} \Rightarrow \boxed{\begin{array}{l} y \\ woman(y) \\ loves(y, john) \end{array}} \end{array}}$$

$$= \Big(\forall x \cdot man(x) \Rightarrow sleeps(x)\Big) \Rightarrow \Big(\exists y \cdot woman(y) \wedge loves(y, john)\Big)$$

Dit is de vertaling zoals we die zouden verwachten.

4.5 Evaluatie

Het framework dat Blackburn en Bos voorstellen, is uitermate geschikt voor semantische analyse van natuurlijke taal. De verschillende onderdelen staat vrij los van elkaar. Zo kan men de doeltaal vrij kijzen. Blackburn en Bos vertalen in hun eerste boek [5] naar eerste-orde-logica. In hun tweede boek [6] vertalen ze naar *Discourse Representation Structures*. Ook de vorm van de grammatica is vrij te kiezen. Zowel Blackburn en Bos als deze thesis gebruiken DCG’s om de grammatica te specifiëren. Baral et al. [3] gebruiken een gelijkaardig framework maar met behulp van een Combinatorische Categorische Grammatica.

Dankzij het framework van Blackburn en Bos, wordt het probleem van semantische analyse herleid tot het opstellen van een lexicon dat de woorden van het logigram bevat; het opstellen van een grammatica die de zinsstructuren van een logigram omvat; het opstellen van de semantiek van deze grammaticale regels; en ten slotte het opstellen van semantische macro’s voor alle lexicale categorieën die we introduceren.

Hoofdstuk 5

Een lexicon voor logigrammen

In dit hoofdstuk bespreken we de lexicale categorieën die gebruikt kunnen worden voor het vertalen van logigrammen naar Discourse Representation Structures. We bespreken zowel de categorieën zelf alsook hun vertaling. Er wordt een onderscheid gemaakt tussen open en gesloten lexicale categorieën. De open categorieën zijn open voor uitbreiding. De woorden uit die categorieën zijn verschillend per logigram. De gesloten categorieën bevatten woorden die gemeenschappelijk zijn voor alle logigrammen. Tabel 5.1 geeft een overzicht van de gebruikte lexicale categorieën.

Sommige van deze categorieën zijn gebaseerd op lexicale categorieën uit de code van Blackburn en Bos [6]. Voor deze categorieën werd de semantiek (grotendeels) overgenomen van Blackburn en Bos. We voegen echter ook een aantal nieuwe categorieën toe en passen de semantiek van sommige categorieën aan.

5.1 Determinator

Een determinator kan zowel een lidwoord als een kwantor zijn. In het geval van logigrammen volstaan de lidwoorden “a”, “an” en “the”. Deze drie determinatoren krijgen alle drie de vertaling van de existentiële determinator uit het vorige hoofdstuk (met R de *restriction* afkomstig van het zelfstandig naamwoord en S de *nuclear scope* of de eigenschap waaraan de naamwoordgroep moet voldoen)

$$[\![det]\!] = [\![det_{existentieel}]\!] = \lambda R \cdot \lambda S \cdot \left(\begin{array}{|c|} \hline x \\ \hline \\ \hline \end{array} \oplus R(x) \oplus S(x) \right)$$

Er is dus geen nood aan een universele of negatieve determinator voor de logigrammen die wij bestudeerden. Bij logigrammen zijn we namelijk op zoek naar de waarde van bijecties. Er is dus altijd exact één iemand die een bepaalde drank drinkt of een bepaald huisdier heeft. Er is nooit sprake van “Every man who drinks vodka, ...” maar altijd van “The man who drinks vodka, ...”. Ook de negatieve determinator (bv. “No man drinks vodka”) wordt nooit gebruikt.

5.2 Hoofdtelwoord

Hoofdtelwoorden zijn determinatoren die een aantal uitdrukken. In deze thesis krijgen hoofdtelwoorden echter een eigen lexicale categorie omdat op sommige plaatsen enkel een hoofdtelwoord past en geen andere determinatoren (bijvoorbeeld “10” in de zin “John is

Categorie	Afkorting	Open?	Voorbeeld
Determinator	det	gesloten	a, an, the
Hoofdtelwoord	number	gesloten	three, 5
Eigennaam	pn	open	John, “the black darts”
Substantief	noun	open	man, year,
Voorzetsel	prep	gesloten	in, to
Betrekkelijk voornaamwoord	relpro	gesloten	who, which, that
Transitief werkwoord	tv	open	loves, “had a final score of”
Hulpwerkwoord	av	gesloten	does, “doesn’t”
Koppelwerkwoord	cop	gesloten	is, “is not”
Comparatief	comp	open	below, “older than”
Onbepaalde woorden	some	open	somewhat, sometime
Voegwoord	coord	gesloten	and, or, “neither ... nor ...”

Tabel 5.1: Een overzicht van de lexicale categorieën

10 years older than Mary”). De hoofdtelwoorden mogen in cijfers voorkomen maar ook in woorden[1].

De signatuur van een hoofdtelwoord is gelijk aan die van een determinator. Er zijn twee mogelijke lezingen voor een hoofdtelwoord: de collectieve en de distributieve lezing. We verduidelijken aan de hand van de voorbeeldzin “Twee mannen gaan naar zee”. In de collectieve lezing vormen de “twee mannen” één geheel. Ze gaan dus samen naar zee. In de distributieve lezing zijn er twee mannen die elk naar zee gaan. In logigrammen gebruiken we enkel de collectieve lezing[2]. Wat ons vooral interesseert is het aantal. Meestal gaat het immers om een numerieke eigenschap van iets of iemand. Bijvoorbeeld “John is 10 years old” of “John is 3 years younger than Mary”.

De formule lijkt heel sterk op die van een determinator maar i.p.v. een existentieel gebonden x wordt nu een getal gebruikt.

$$[\![number]\!] = \lambda R \cdot \lambda S \cdot \{R\,(Number) \oplus S\,(Number)\}$$

5.3 Eigennaam

Een eigennaam is een open lexicale categorie. Dat wil zeggen dat de eigennamen verschillend zijn per logigram. De semantiek is identiek aan die in het vorige hoofdstuk.

$$[\![pn]\!] = \lambda E \cdot E\,(Symbool)$$

Een eigennaam voldoet aan een eigenschap E als de constante die geïntroduceerd wordt door de eigennaam, de functiewaarde *waar* heeft voor de functie E.

We staan toe dat woordgroepen die taalkundig geen eigennaam zijn, toch gebruikt kunnen worden als een eigennaam. Zo kan “the black darts” (wat normaal een determinator + adjectief + substantief is) aanzien worden als een eigennaam. Dit maakt het vertalen van de zinnen makkelijker maar tegelijkertijd wordt het opstellen van het lexicon voor een logigram moeilijker. Het lexicon is niet meer enkel afhankelijk van taalkundige informatie. Alle mogelijke domeinelementen van een (niet-numeriek) domein

[1] In de praktijk zitten enkel de eerste 15 getallen in woorden in het lexicon

[2] De distributieve lezing wordt heel af en toe gebruikt maar wordt niet ondersteund door onze grammatica

moeten namelijk als een eigennaam aangegeven worden in het lexicon dat hoort bij het logigram. Bovendien moet dit gebeuren in de vorm zoals het voorkomt in de zinnen van het logigram. Zo is "black" een waarde van het concept kleur, toch moet "the black darts" ingegeven worden in het lexicon. Deze eigennamen zullen later vertaald worden naar een constante uit een constructed type (zie ook hoofdstuk 8). Daarmee is het lexicon dus een mengeling van een taalkundig en een formeel vocabularium.

Een logigram kan 3 soorten eigennamen hebben: een eigennaam in het enkelvoud (bv. "John"), een eigennaam in het meervoud (bv. "The Turkey Rolls") en een *numerieke eigennaam* (bv. "March"). Bij de eerste twee wordt het symbool afgeleid van de woordvorm. Bij de laatste gebeurt dit door de gebruiker. Numerieke eigennamen worden namelijk gebruikt om woorden om te zetten in getallen. Zo kan "March" omgezet worden in 3. Op die manier wordt het zinvol om te spreken over "1 maand na maart". Voor een *numerieke eigennaam* is het symbool gelijk aan die numerieke waarde. Deze wordt apart meegegeven in het probleem-specifiek lexicon. De vertaling van woorden naar getallen moet door de gebruiker gebeuren omdat hier achtergrondkennis voor nodig is. Het is opnieuw een voorbeeld van hoe onze vertaling van een logigram deels in het lexicon zit.

5.4 Substantief

Ook substantieven zijn een open categorie. Hun semantiek nemen we voorlopig over van het vorige hoofdstuk.

$$[\![noun]\!] = \lambda x \cdot \boxed{\begin{array}{|c|}\hline \\ \hline Symbool(x) \\ \hline\end{array}}$$

In hoofdstuk 7 (over types) zullen we DRS uitbreiden met types en zal het predicaat op x verdwijnen en vervangen worden door een echte type-constraint in een DRS. Een substantief in het logigram-specifiek lexicon bevat een enkelvoudsvorm en een meervoudsvorm. Het symbool is gelijk aan de enkelvoudsvorm. Op die manier hebben de enkelvoudsvorm en de meervoudsvorm dezelfde waarde voor de feature `Symbool`.

5.5 Voorzetsel

De voorzetsels (in het Engels `prepositions` of `prep`) vormen een gesloten woordklasse die bestaat uit woorden zoals "from", "in" en "with". Ze worden op twee manieren gebruikt in de zinnen van een logigram. Enerzijds kan het voorzetsel gebruikt worden bij een werkwoord. Dan staat het voorzetsel vlak voor het lijdend voorwerp. Bijvoorbeeld de "with" in "to finish with 500 points". Anderzijds is een voorzetsel het belangrijkste woord in een voorzetselconstituent (ook wel `prepositional phrase` of `pp` genoemd). Bijvoorbeeld de "from" in "the man from France". Het voorzetsel dat bij een werkwoord hoort, zien we als deel van het werkwoord. In dat geval heeft het voorzetsel geen vertaling.

In het geval van een voorzetselconstituent is er wel een vertaling. We kunnen zo'n voorzetselconstituent zien als een extra beperking op een substantief. Of een transformatie van een substantief naar een nieuw substantief. Met τ de signatuur van een woordgroep wordt dit $\tau(pp) = \tau(n) \to \tau(n)$. Een voorzetsel is dan weer een functie van een naamwoordgroep (`noun phrase` of `np`) naar een voorzetselconstituent. Of in formulevorm $\tau(prep) = \tau(np) \to \tau(pp) = \tau(np) \to \tau(n) \to \tau(n) = [(e \to t) \to t] \to (e \to t) \to (e \to t)$.

De betekenis ziet er uit als

$$[\![prep]\!] = \lambda N \cdot \lambda S \cdot \left(\lambda y \cdot \left\{ S\left(y\right) \oplus N \left(\lambda x \cdot \boxed{\frac{\quad}{Symbool(y,x)}} \right) \right\} \right)$$

Een voorzetsel neemt een naamwoordgroep N en een substantief S als argument en geeft een beperking op een entiteit y terug. De beperking op y bestaat enerzijds uit de beperking van het substantief, namelijk $S\left(y\right)$. Anderzijds voegt het voorzetsel zelf ook nog een beperking toe. De naamwoordgroep N moet namelijk voldoen aan de eigenschap $\lambda x \cdot \boxed{\frac{\quad}{Symbool(y,x)}}$, bijvoorbeeld voor "with" $\lambda x \cdot \boxed{\frac{\quad}{with(y,x)}}$.

Daardoor zal een voorzetselconstituent zoals "with the black darts" de betekenis

$$[\![pp]\!] = \lambda S \cdot \left(\lambda y \cdot \left\{ S\left(y\right) \oplus \boxed{\frac{\quad}{with(y, TheBlackDarts)}} \right\} \right)$$

krijgen

5.6 Betrekkelijk voornaamwoord

Een betrekkelijk voornaamwoord (`relative pronoun` of `relpro`) is een woord aan het begin van een betrekkelijke bijzin (ook wel `relative clause` of `rc`). Voorbeelden zijn "that", "which" en "who". Net als een voorzetselconstituent staat zo'n betrekkelijke bijzin bij een substantief en legt ze een extra beperking op.

$$[\![relpro]\!] = \lambda V \cdot \lambda S \cdot (\lambda x \cdot \{S\left(x\right) \oplus V\left(\lambda E \cdot E\left(x\right)\right)\})$$

Een betrekkelijk voornaamwoord neemt een verbale constituent V en een substantief S als argument en geeft een beperking op een entiteit x terug. Die beperking op x bestaat uit een conjunctie ($\oplus$-operator) van de beperking van het substantief S (namelijk $S\left(x\right)$) en van een beperking van de verbale constituent. De entiteit x moet namelijk ook voldoen aan de eigenschap E van de verbale constituent V.

De x is gebonden door de lambda-functie en dus binnen de lambda-expressie eenduidig bepaald. Men kan $\lambda E \cdot E\left(x\right)$ daarom ook zien als een soort van eigennaam x die het onderwerp vormt van de verbale constituent V.

5.7 Transitief werkwoord

Een transitief (of overgankelijk) werkwoord is een werkwoord met een lijdend voorwerp. Een logigram heeft enkel transitieve werkwoorden. Er zijn geen intransitieve (zonder lijdend voorwerp) of ditransitieve werkwoorden (met een meewerkend voorwerp). De vertaling van een transitief werkwoord is gelijk aan die van het vorige hoofdstuk

$$[\![tv]\!] = \lambda L \cdot \lambda O \cdot O \left(\lambda x_o \cdot L \left(\lambda x_l \cdot \boxed{\frac{\quad}{Symbool(x_o, x_l)}} \right) \right)$$

Een entiteit x_o omschreven door het onderwerp O voldoet aan de verbale constituent als voor die x_o het lijdend voorwerp L voldoet aan de eigenschap $\lambda x_l \cdot \boxed{\frac{\quad}{Symbool(x_o, x_l)}}$. We passen dit toe op "A man loves every woman". Een man x_o voldoet aan de verbale

constituent (“loves every woman”) als voor die man x_o en elke vrouw x_l geldt dat $\boxed{\begin{array}{|c|}\hline \\ \hline loves(x_o, x_l)\\ \hline\end{array}}$ waar is. Het onderwerp (als geheel) voldoet aan de verbale constituent als er zo’n man x_o bestaat.

Een transitief werkwoord is een open lexicale categorie. Dat wil zeggen dat de woorden verschillend zijn per logigram. In het logigram-specifiek lexicon staat de infinitief, de werkwoordsvorm in de derde persoon enkelvoud alsook een voltooid of onvoltooid deelwoord. De enkelvoudsvorm kan zowel in de verleden tijd als de tegenwoordige tijd zijn, afhankelijk van hoe het werkwoord gebruikt wordt in het logigram. Ten slotte wordt ook nog het voorzetsel gegeven dat voor het lijdend voorwerp wordt gezet (indien van toepassing) en eventueel een achtervoegsel aan het einde van de zin (bv. “to print” in “The design took 8 minutes to print”). Het symbool van het werkwoord (en dus ook de naam van het predicaat) wordt afgeleid uit de enkelvoudsvorm, het voorzetsel en het achtervoegsel.

Net zoals met de eigennamen wordt er vrij los omgesprongen met de werkwoorden. Zo is “to be recognized as endangered in” een werkwoord in één van de logigrammen.

5.8 Hulpwerkwoord

De woordklasse van hulpwerkwoorden is een gesloten woordklasse. Binnen de logigrammen zijn het de werkwoordsvormen van “to do” en “to be” die deel uitmaken van deze klasse. Alsook het hulpwerkwoord voor de toekomst “will”. Naast de woordvorm bevat het lexicon ook informatie over de polariteit van die woordvorm: positief of negatief. “does” is positief, “doesn’t” is negatief. De beide polariteiten hebben elk een andere betekenis.

Een hulpwerkwoord is een woord dat een verbale constituent (`verb phrase` of `vp`) omvormt tot een nieuwe verbale constituent. Voor een hulpwerkwoord met positieve polariteit is dit de identieke transformatie[3]. Dit is bijvoorbeeld het geval voor “is” in “John is cleaning the house.”.

$$[\![av_{pos}]\!] = \lambda V \cdot V$$

Voor een hulpwerkwoord met een negatieve polariteit bestaat de transformatie uit een negatie van de verbale constituent. Er is dus geen negatie van het onderwerp. Hiermee wordt de vertaling van “Everyone doesn’t work” naar logica $\forall x \cdot \neg work(x)$ i.p.v. $\neg\forall x \cdot work(x)$

$$[\![av_{neg}]\!] = \lambda V \cdot \lambda O \cdot O\left(\lambda x_o \cdot V\left(\lambda E \cdot \boxed{\begin{array}{|c|}\hline \\ \hline \neg E\,(x_o)\\ \hline\end{array}}\right)\right)$$

Het hulpwerkwoord krijgt een verbale constituent (V) en een onderwerp (O) als argument. Een entiteit x_o omschreven door het onderwerp voldoet aan de verbale constituent inclusief hulpwerkwoord als het niet voldoet aan de eigenschap E van de verbale constituent V die deel uitmaakt van de gehele verbale constituent.

[3] De tijd maakt niet uit voor een logigram. “John will clean the house” moet dus niet anders vertaald worden dan “John cleans the house”

5.9 Koppelwerkwoord

De categorie van koppelwerkwoorden is een gesloten lexicale categorie. Ze bestaat uit verschillende vormen van het werkwoord "to be" (bv. is, isn't, is not, was, are, were, ...). Er is een enkelvoud- en meervoudsvorm. Bovendien is er sprake van een positieve of negatieve polariteit. Deze hebben elk een licht andere semantiek. Ten slotte kan een koppelwerkwoord ook op drie verschillende manieren gebruikt worden in een zin van een logigram:

- Samen met een `nominale constituent` (`noun phrase` of `np`): Bijvoorbeeld "John is a man". Dit type van gebruik heeft dezelfde signatuur als een overgankelijk werkwoord. De semantiek zegt dat de een entiteit x_o van het onderwerp O voldoet als het lijdend voorwerp L voldoet aan $\lambda x_l \cdot \boxed{\begin{array}{|c|}\hline \\ \hline x_o = x_l \\ \hline\end{array}}$. Voor "John is a man" wilt dit zeggen dat er een man x_l moet zijn die voldoet aan de DRS-conditie $x_l = John$. Binnen logigrammen zijn de naamwoordgroep van het onderwerp en het lijdend voorwerp altijd volledig gespecificeerd. De semantiek wil dan zeggen dat deze twee entiteiten gelijk zijn aan elkaar.

$$[\![cop_{np,pos}]\!] = \lambda L \cdot \lambda O \cdot O \left(\lambda x_o \cdot L \left(\lambda x_l \cdot \boxed{\begin{array}{|c|}\hline \\ \hline x_o = x_l \\ \hline\end{array}} \right) \right)$$

  In de negatieve vorm, is er ook een negatie van het lijdend voorwerp. Zodanig dat er een correcte negatie van de kwantoren is. In de zin "Mary is not a man" is het belangrijk dat er geen enkele man is die gelijk is aan Mary. De negatie moet dus voor de existentiële kwantor komen en dus voor het lijdend voorwerp.

$$[\![cop_{np,neg}]\!] = \lambda L \cdot \lambda O \cdot O \left(\lambda x_o \cdot \boxed{\begin{array}{|l}\hline \\ \neg \left[L \left(\lambda x_l \cdot \boxed{\begin{array}{|c|}\hline \\ \hline x_o = x_l \\ \hline\end{array}} \right) \right] \\ \hline\end{array}} \right)$$

- Samen met een `adjectiefconstituent` (`adjective phrase` of `ap`): Bijvoorbeeld "John is 30 years old". De vertaling die wij gebruiken ligt echter vrij ver van de taalkundige structuur. Zo wordt het koppelwerkwoord + adjectief gezien als een soort van transitief werkwoord. De betekenis is gelijk aan die van het transitieve werkwoord en voor de negatie gelijkaardig aan die van het koppelwerkwoord met een nominale constituent. De adjectieven die gebruikt kunnen worden, moeten meegegeven worden via het lexicon van een logigram. Het symbool komt overeen met de woordvorm van het adjectief.

$$[\![cop_{ap,pos}]\!] = [\![tv]\!] = \lambda L \cdot \lambda O \cdot x_o \left(\lambda x_o \cdot L \left(\lambda x_l \cdot \boxed{\begin{array}{|c|}\hline \\ \hline Symbool(x_o, x_l) \\ \hline\end{array}} \right) \right)$$

$$[\![cop_{ap,neg}]\!] = \lambda L \cdot \lambda O \cdot O \left(\lambda x_o \cdot \boxed{\begin{array}{|l}\hline \\ \neg \left[L \left(\lambda x_l \cdot \boxed{\begin{array}{|c|}\hline \\ \hline Symbool(x_o, x_l) \\ \hline\end{array}} \right) \right] \\ \hline\end{array}} \right)$$

- Samen met een `voorzetselconstituent` (`prepositional phrase` of `pp`): Bijvoorbeeld "John is from France". Dit kan aanzien worden als een ellips van "a person":

"John is a person from France". De signatuur is $\tau(cop_{pp}) = \tau(pp) \to \tau(vp) = \tau(pp) \to \tau(np) \to t = [(e \to t) \to e \to t] \to [(e \to t) \to t] \to t$.

$$[\![cop_{pp,pos}]\!] = \lambda V \cdot \lambda O \cdot O \left(\lambda x_o \cdot V \left(\lambda y \cdot \boxed{\begin{array}{|l} \\ \hline \\ \end{array}} \right) (x_o) \right)$$

Het koppelwerkwoord krijgt een voorzetselconstituent V en het onderwerp O als argument. Een entiteit x_o van het onderwerp voldoet aan de verbale constituent als het voldoet aan de eigenschap van de voorzetselconstituent V. Die voorzetselconstituent krijgt als eerste argument nog een een lambda-functie mee die een beperking oplegt vanuit het substantief waarbij het hoort. In dit geval is er echter geen substantief en dus geen beperking. Dit vertaalt zich in een lege DRS-structuur. Dit is equivalent aan $\lambda y \cdot true$ in eerste-orde-logica.
De betekenis van de negatieve vorm is

$$[\![cop_{pp,neg}]\!] = \lambda V \cdot \lambda O \cdot O \left(\lambda x_o \cdot \boxed{\begin{array}{|l} \\ \hline \neg \left[V \left(\lambda y \cdot \boxed{\begin{array}{|l} \\ \hline \\ \end{array}} \right) (x_o) \right] \\ \end{array}} \right)$$

5.10 Comparatief en onbepaalde woorden

Binnen een logigram is er vaak een domein van numerieke aard. In dat geval is er meestal ook een zin die uitdrukt dat iemand een hogere of lagere numerieke waarde heeft dan iemand anders. Bijvoorbeeld "John scored 15 points less than Mary" of "Mary finished sometime after Tom". De woordgroepen "less than" en "after" zijn deel van de lexicale categorie `comparatief`. "sometime" is dan weer een voorbeeld van een onbepaald woord. Wat op het eerste zicht niet opvalt aan deze zinnen is dat ze een belangrijke ellips bevatten. Tom is namelijk geen tijdstip maar een persoon. De tweede zin is voluit "Mary finished sometime after Tom finished". Deze ellips zal opgelost worden in de grammatica.

Er zijn opnieuw 2 betekenissen afhankelijk van het woord: één voor "hoger" (bv. "after") en één voor "lager" (bv. "less than")

$$[\![comp_{lager}]\!] = \lambda H \cdot \lambda N \cdot \left(\lambda E \cdot H \left(\lambda x_h \cdot N \left(\lambda x_n \cdot \left\{ \boxed{\begin{array}{|l} x \\ \hline x = x_n - x_h \end{array}} \oplus E\,(x) \right\} \right) \right) \right)$$

Een comparatief neemt twee naamwoordgroepen als argument. De eerste is een hoeveelheid H en de tweede een numerieke eigenschap N. Het resultaat is een nieuwe naamwoordgroep die kan dienen als lijdend voorwerp. Een naamwoordgroep is van type $(e \to t) \to t$ en dus krijgen we nog een argument E van type $e \to t$ mee. Dit is de eigenschap waaraan de nieuwe naamwoordgroep moet voldoen. De semantiek zegt dan dat de naamwoordgroepen H en N respectievelijk een entiteit x_h en x_n moeten omschrijven zodanig zijn dat er een geheel getal $x = x_n - x_h$ bestaat dat voldoet aan de eigenschap E. Zo geldt voor "John scored 15 points less than Mary scored" dat $x_h = 15$, x_n is de score van Mary en $x = x_n - 15$ moet voldoen aan de eigenschap $\lambda x \cdot \boxed{\begin{array}{|l} \\ \hline scored(John, x) \end{array}}$.

De andere betekenis van de comparatief, die voor "hoger", is gelijkaardig aan de eerste

$$[\![comp_{hoger}]\!] = \lambda H \cdot \lambda N \cdot \left(\lambda E \cdot H \left(\lambda x_h \cdot N \left(\lambda x_n \cdot \left\{ \boxed{\begin{array}{|l} x \\ \hline x = x_n + x_h \end{array}} \oplus E\,(x) \right\} \right) \right) \right)$$

De comparatief is een open lexicale categorie. Elke logigram heeft eigen comparatieven. In het logigram-specifiek lexicon bevindt zich naast de woordvorm ook telkens het type ("hoger" of "lager").

De onbepaalde woorden hebben dezelfde signatuur als een naamwoordgroep $\tau(some) = \tau(np) = (e \to t) \to t$. Een onbepaald woord voldoet aan een eigenschap E als er een natuurlijk getal bestaat dat voldoet aan die eigenschap[4]. Meer concreet hebben we

$$[\![some]\!] = \lambda E \cdot \left\{ \begin{array}{|c|}\hline x \\ \hline x > 0 \\ \hline \end{array} \oplus E\,(x) \right\}$$

5.11 Voegwoord

Een voegwoord is een woord dat twee woordgroepen van dezelfde categorie verbindt. Voorbeelden zijn "and", "or" and "nor". Een voegwoorden kan twee zinnen verbinden maar ook twee nominale constituenten. Binnen logigrammen zijn de voegwoorden nevenschikkend. Dat wil zeggen dat beide woordgroepen even belangrijk zijn[5]. Er komen drie soorten voegwoorden voor, elk met een eigen vertaling: een conjunctief voegwoord ("A and B"), een disjunctief voegwoord ("A or B") en een negatief voegwoord ("A nor B"). Een voegwoord kan uit twee delen bestaan (bv. "either ... or" en "neither ... nor"). Daarom bestaan er twee lexicale categorieën: `coord` voor het voegwoord zelf (bv. "or") en `coordPrefix` voor de eventuele prefix (bv. "either").

De vertaling van het voegwoord is onafhankelijk van de categorie van de woordgroepen die worden verbonden. Belangrijk is wel dat de signatuur van die woordgroepen een functie met als resultaat iets van type t is $(\tau(woordgroep) = \alpha \to t)$. Een signatuur van het voegwoord is dan $\tau(coord) = (\alpha \to t) \to (\alpha \to t) \to (\alpha \to t) = (\alpha \to t) \to (\alpha \to t) \to \alpha \to t$. De betekenis van een voegwoord bestaat er telkens uit door het derde argument (van type α) door te geven aan de eerste twee argumenten en de resultaten op de juiste manier terug te combineren[6].

$$[\![coord_{conjunctief}]\!] = \lambda A \cdot \lambda B \cdot (\lambda C \cdot \{A\,(C) \oplus B\,(C)\})$$

$$[\![coord_{disjunctief}]\!] = \lambda A \cdot \lambda B \cdot \left(\lambda C \cdot \begin{array}{|c|}\hline \\ \hline A\,(C) \vee B\,(C) \\ \hline \end{array} \right)$$

$$[\![coord_{negatief}]\!] = \lambda A \cdot \lambda B \cdot \left(\lambda C \cdot \left\{ \begin{array}{|c|}\hline \\ \hline \neg A\,(C) \\ \hline \end{array} \oplus \begin{array}{|c|}\hline \\ \hline \neg B\,(C) \\ \hline \end{array} \right\} \right)$$

Deze vertaling is echter niet de enige mogelijke vertaling. In het geval van een disjunctie kan men ook naar een exclusieve of vertalen (zeker voor de constructie "either ... or ..." houdt dit steek). Omwille van de bijecties bij logigrammen maakt dit echter geen verschil. Er is immers altijd maar exact één iemand die eraan voldoet dus er is in praktijk geen verschil tussen een inclusieve en exclusieve disjunctie.

[4] Het getal moet strikt positief zijn want anders kan $x = x2 - x1$ ook hoger dan $x2$ uitkomen bij een comparatief met als betekenis "lager dan"

[5] In tegenstelling tot een onderschikkend voegwoord. Zo'n voegwoord verbindt bijvoorbeeld een hoofdzin met een bijzin. Daarbij is de bijzin minder belangrijk dan de hoofdzin.

[6] $A\,(C)$ en $B\,(C)$ zijn van type t en dus DRS-structuren

Echter ook voor de conjunctie is een andere vertaling mogelijk, meer bepaald voor de conjunctie van twee naamwoordgroepen. Zo heeft de zin “John and Mary went to the sea” twee mogelijke vertalingen. De bovenstaande vertaling wordt de distributieve lezing genoemd en komt overeen met dat beiden (apart) naar de zee zijn gegaan. In de andere vertaling, de collectieve lezing, zijn John en Mary samen naar de zee gegaan. Binnen logigrammen komt de collectieve lezing bijna niet aan bod. De enige plaats waar dat wel gebeurt, is in een soort van *alldifferent*-constraint. Bijvoorbeeld “John, Bob and Charles are three different politicians”. Bij de vertaling van de grammatica lossen we dit probleem op.

5.12 Conclusie

We hebben twaalf lexicale categorieën opgesteld voor het vertalen van logigrammen naar logica. Een aantal van deze categorieën zijn specifiek aan logigrammen (bijvoorbeeld de onbepaalde woorden). De meeste zijn echter taalkundig verantwoord.

We springen wel vrij los om met welke woorden behoren tot de lexicale categorieën. Daardoor wordt het lexicon een mengeling van een taalkundig en formeel vocabularium. Zo moeten alle niet-numerieke domeinelementen voorkomen als eigennaam in het lexicon. Bovendien moeten numerieke eigennamen (zoals “March”) een getal meekrijgen dat gebruikt wordt in de vertaling.

Aangezien de betekenis van een zin hoofdzakelijk in de betekenis van de woorden zit, bevat de semantiek van het lexicon ook een aantal keuzes over welke betekenis gebruikt wordt voor het vertalen van logigrammen naar logica. Zo impliceert onze semantiek van een transitief werkwoord dat de kwantoren van het onderwerp voor die van het lijdend voorwerp moeten komen. Binnen logigrammen volstaat deze betekenis. Bij uitbreiding naar andere domeinen moeten deze keuzes opnieuw geëvalueerd worden om te zien of ze nog altijd van toepassing zijn.

Hoofdstuk 6

Een grammatica voor logigrammen

In dit hoofdstuk overlopen we de grammaticale categorieën voor de vertaling van logigrammen naar logica. We bespreken ook telkens de grammaticale regels horend bij die categorie en hun semantiek. We beginnen bij de categorieën die bestaan uit lexicale categorieën en werken naar boven toe richting de categorie van een zin. We gebruiken de DCG-notatie uit hoofdstuk 1.1. De feature `sem` wordt geünificeerd met de semantiek van een woordgroep.

De grammatica is opgesteld vertrekkend van een grammatica uit de code van Blackburn en Bos [6]. Deze code werd aangepast aangepast om de eerste 10 logigrammen uit Puzzle Baron's Logic Puzzles Volume 3 [20] te kunnen vertalen. De grammatica regels van Blackburn en Bos die niet van toepassing zijn voor logigrammen zijn verwijderd.

6.1 De link met het lexicon

In deze sectie bespreken we de grammticale categorieën die overeenkomen met een lexicale categorie.

6.1.1 Een mapping naar het lexicon

De meeste van de grammaticale categorieën die de link met het lexicon vormen, zijn een directe mapping naar dit lexicon. Een woord(groep) behoort tot een grammaticale categorie als het tot de overeenkomstige lexicale categorie behoort. De features van de grammaticale categorie komen overeen met die van de lexicale categorie. De semantiek van de woordgroep komt overeen met de formules uit hoofdstuk 5.

De categorieën die zo'n directe link vormen zijn determinatoren (`det`), hoofdtelwoorden (`number`), substantieven (`noun`), voorzetsels (`prep`), hulpwerkwoorden (`av`), koppelwerkwoorden (`cop`) en comparatieven (`comp`).

6.1.2 Eigennaam

De grammaticale categorie van eigennamen verschilt lichtjes van de overeenkomstige lexicale categorie. Voor de grammaticale categorie kan een eigennaam uit het lexicon namelijk voorafgegaan zijn door "the". Deze optionele "the" is nodig omdat in sommige puzzels een bepaalde term zowel met als zonder "the" voorkomt. Indien we deze twee

verschillende termen allebei apart in het lexicon zouden ingeven, zouden deze worden vertaald naar verschillende symbolen. Dit zouden we graag vermijden[1].

6.1.3 Transitief werkwoord

Er zijn twee grammaticale regels voor een transitief werkwoord. Enerzijds is er de standaard mapping van grammatica naar lexicon. De voorzetsels en achtervoegsels uit het lexicon worden dan als feature meegegeven aan de grammaticale woordgroep. Anderzijds wordt de combinatie koppelwerkwoord + adjectief ook als een transitief werkwoord gezien, bijvoorbeeld "is old" in "John is 18 years old". Hierbij is het adjectief het achtervoegsel. De semantiek van deze combinatie is die van het koppelwerkwoord ($[\![cop_{ap}]\!]$) zoals we die hebben afgeleid in hoofdstuk 5.9.

6.1.4 Onbepaalde woorden en betrekkelijke voornaamwoorden

Het is mogelijk om een onbepaald woord te verzwijgen, bijvoorbeeld "John finished [sometime] before Mia". Daarom is er een extra regel die zegt dat het woord niet per se hoeft voor te komen in de zin. Hetzelfde geldt voor betrekkelijke voornaamwoorden.

6.1.5 Voegwoord

Voegwoorden kunnen uit twee delen bestaan: het voegwoord zelf (`coord`) en een optionele prefix (`coordPrefix`). De prefix heeft geen vertaling bovenop het voegwoord. De grammaticale regel voor het voegwoord zelf volgt dezelfde structuur als die van sectie 6.1.1. Een voegwoord kan soms echter weggelaten worden. De grammaticale categorie `noCoord` stelt een ellips van zo'n voegwoord voor. Er is gekozen voor een aparte categorie omdat een ellips niet overal mag voorkomen.

6.2 (Getransformeerde) Substantieven

Grammatica 6.1 geeft een overzicht van de grammatica regels i.v.m. substantieven en transformaties op die substantieven. De eerste 2 regels zeggen dat een getransformeerd substantief (hiervoor gebruiken we de categorie `n`) kan bestaan uit een gewoon substantief (`noun`) al dan niet gevolgd door een transformatie op dat substantief (`nmod` naar noun modifier). In het geval van een transformatie is de betekenis van het getransformeerde substantief gelijk aan de applicatie van de transformatie op dat van het echte substantief of $[\![n]\!] = [\![nmod]\!]\,([\![noun]\!])$.

Lijnen 9 t.e.m. 14 van grammatica 6.1 definiëren de mogelijke transformaties: ofwel gaat het om een voorzetselconstituent (`pp`), ofwel om een betrekkelijke bijzin (`rc`). In het laatste geval moet het getal van het substantief en het werkwoord uit de bijzin overeenkomen.

```
1 n([num:Num, sem:Sem]) -->
2   noun([num:Num, sem:Noun]),
3   { Sem = Noun }.
4 n([num:Num, sem:Sem]) -->
5   noun([num:Num, sem:Noun]),
6   nmod([num:Num, sem:NMod]),
```

[1] Een alternatief was om de twee vormen met hetzelfde symbool in het lexicon op te nemen. Dat is equivalent aan de beschreven grammatica

```
7   { Sem = app(NMod, Noun) }.
8
9   nmod([num:_, sem:Sem]) -->
10    pp([sem:PP]),
11    { Sem = PP }.
12  nmod([num:Num, sem:Sem]) -->
13    rc([num:Num, sem:RC]),
14    { Sem = RC }.
15
16  pp([sem:Sem]) -->
17    prep([sem:Prep]),
18    np([coord:_, num:_, gap:[], sem:NP]),
19    { Sem = app(Prep, NP) }.
20
21  rc([num:Num, sem:Sem]) -->
22    relpro([sem:RelPro]),
23    vp([coord:no, inf:fin, num:Num, gap:[], sem:VP]),
24    { Sem = app(RelPro, VP) }.
```

Grammatica 6.1: De grammaticale regels i.v.m. substantieven

Lijnen 16 t.e.m. 19 beschrijven de voorzetselconstituent (`pp`) zoals "from France". Deze constituent bestaat namelijk uit een voorzetsel (`prep`) gevolgd door een nominale constituent (`np`). De feature `gap` van de nominale constituent heeft te maken met ellipsen die kunnen voorkomen in die constituent. De lege lijst wilt zeggen dat er geen ellipsen mogelijk zijn.

De betekenis van een voorzetselconstituent bestaat uit een simpele lambda-applicatie ($[\![pp]\!] = [\![prep]\!]\,([\![np]\!])$).

Lijnen 21 t.e.m. 24 beschrijven de betrekkelijke bijzin (`rc`). Zo'n betrekkelijke bijzin bestaat uit een betrekkelijk voornaamwoord (`relpro`) gevolgd door een verbale constituent. Een voorbeeld van zo'n betrekkelijke bijzin is "that loves Mary". De verbale constituent moet simpel zijn (`coord:no`). Dat wil zeggen dat er geen voegwoorden mogen in voorkomen. Bovendien moet het werkwoord vervoegd zijn (`inf:fin`) en dus niet in de infinitief voorkomen. Ten slotte moet het getal overeenkomen met dat van het substantief.

De betekenis bestaat opnieuw uit één lambda-applicatie, namelijk $[\![rc]\!] = [\![relpro]\!]\,([\![vp]\!])$.

6.3 Nominale constituent

Een nominale constituent of naamwoordgroep (`np`) is een woordgroep waarin het naamwoord het belangrijkste woord is en die een entiteit aanduidt. Deze woordgroepen kunnen dienen als onderwerp of als lijdend voorwerp van een werkwoord.

De grammaticale categorie heeft 4 mogelijke features: De feature `coord` duidt aan of het gaat om een simpele naamgroep (`coord:no`) of een complexere naamgroep. Zo heeft een vergelijkende naamwoordgroep de waarde `comp` voor deze feature. De feature `num` duidt het getal van de woordgroep aan (enkelvoud `sg` of meervoud `pl`). De feature `gap` geeft aan welke woorden gebruikt zijn in een ellips binnen deze woordgroep. De waarde `[]`

voor deze feature wilt bijvoorbeeld zeggen dat er geen ellips zit in deze naamwoordgroep. Zoals altijd zit de semantiek in de feature `sem`.

6.3.1 Een eenvoudige naamwoordgroep

Grammatica 6.2 geeft de eenvoudige naamwoordgroepen weer. De eerste grammaticale regel stelt dat een naamwoordgroep kan bestaan uit een eigennaam (`pn`) met hetzelfde getal. In dit geval is de semantiek gelijk aan die van de naamwoordgroep.

Daarnaast kan een naamwoordgroep ook bestaan uit een lidwoord (of meer algemeen, een determinator `det`) en een zelfstandig naamwoord. We gebruiken hier de categorie `n` i.p.v. `noun` zodat ook een zelfstandig naamwoord met een bijzin mogelijk is. De semantiek bestaat uit een simpele lambda-applicatie $[\![np]\!] = [\![det]\!]\,([\![n]\!])$.

Zoals we ook in sectie 5.2 besproken hebben, kan een hoofdtelwoord ook aanzien worden als een determinator. We krijgen dus nog een gelijkaardige regel als de vorige maar nu met een hoofdtelwoord i.p.v. een determinator.

Daarnaast kan een getal ook op zichzelf staan zonder zelfstandig naamwoord, bijvoorbeeld als jaartal. De grammaticale regel van lijn 15 t.e.m. 17 dekt dit geval. De semantiek is vergelijkbaar met de vorige grammaticale regel maar er is geen extra restrictie op het getal. Vandaar de lege DRS-structuur in $[\![np]\!] = [\![number]\!]\left(\lambda x \cdot \boxed{\;\;\;\;}\right)$.

Een naamwoordgroep kan vooraf gegaan zijn door een dollar-symbool. Wanneer we types introduceren in hoofdstuk 7 fixeren we het type van deze naamwoordgroep op `dollar`.

Ten slotte kan een naamwoordgroep bestaan uit “whoever” gevolgd door een verbale constituent, bijvoorbeeld “whoever loves Mary” Dit zou voluit neerkomen op “the entity that loves Mary”. Het bestaat dus eigenlijk uit een verzwegen determinator met een ontbrekend substantief en een verzwegen betrekkelijk voornaamwoord. De betekenis is gelijk aan

$$[\![np]\!] = [\![det]\!]\left(([\![relpro]\!]\,(vp))\left(\lambda x \cdot \boxed{\;\;\;\;}\right)\right)$$

Het lidwoord krijgt een getransformeerd substantief mee. Die transformatie bestaat uit een (verzwegen) betrekkelijk voornaamwoord en een verbale constituent. Het substantief dat getransformeerd wordt ontbreekt en legt dus geen restrictie op. Vandaar de lege DRS die overeenkomt met $\lambda x \cdot true$ in eerste-orde-logica.

```
1  np([coord:no, num:Num, gap:[], sem:Sem]) -->
2    pn([num:Num, sem:PN]),
3    { Sem = PN }.
4
5  np([coord:no, num:Num, gap:[], sem:Sem]) -->
6    det([num:Num, sem:Det]),
7    n([num:Num, sem:N]),
8    { Sem = app(Det, N) }.
9
10 np([coord:no, num:Num, gap:[], sem:Sem]) -->
11   number([sem:Number]),
12   n([num:Num, sem:N]),
13   { Sem = app(Number, N) }.
14
15 np([coord:no, num:_, gap:[], sem:Sem]) -->
```

```
16   number([sem:Number]),
17   { Sem = app(Number, lam(_, drs([], []))) }.
18
19 np([coord:no, num:Num, gap:[], sem:Sem]) -->
20   ['$'],
21   np([coord:no, num:Num, gap:[], sem:NP]),
22   { Sem = NP }.
23
24 np([coord:no, num:Num, gap:[], sem:Sem]) -->
25   { semLex(det, [num:Num, sem:Det]) },
26   [whoever],
27   { semLex(relpro, [sem:RP]) },
28   vp([coord:no, inf:fin, num:Num, gap:[], sem:VP]),
29   { Sem = app(Det, app(app(RP, VP), lam(_, drs([], [])))) }.
```

Grammatica 6.2: De grammaticale regels voor een simpele naamwoordgroep

6.3.2 Een naamwoordgroep met onbekende relatie

Grammatica 6.3 beschrijft twee grammaticale regels met dezelfde semantiek en gelijkaardige structuur. Het gaat om naamwoordgroepen respectievelijk zoals “the 2008 graduate” en “John’s dog”. Ze bestaan allebei uit een simpele naamwoordgroep (`np` met `coord:no`) en een substantief (`n`). Er is telkens een link tussen de entiteit van het substantief en dat van de simpele naamwoordgroep maar er is geen kennis over welke link dit juist is. Voor nu laten we het dus nog open. In hoofdstuk 7 zullen we met behulp van types afleiden welk predicaat nodig is. De entiteit van de substantief is het belangrijkste en is ook de entiteit van de naamwoordgroep als geheel. Als resultaat krijgen we

$$[\![np]\!] = \lambda E \cdot ([\![det]\!]\,([\![n]\!])) \left(\lambda x \cdot \left\{ E\,(x) \oplus [\![np]\!] \left(\lambda y \cdot \boxed{\begin{array}{c} \\ \hline unknown(x,y) \end{array}} \right) \right\} \right)$$

Het is een naamwoordgroep en moet dus voldoen aan een eigenschap E. De naamwoordgroep voldoet hier aan als de naamwoordgroep die bestaat uit het lidwoord en het substantief voldoet aan de eigenschap die bestaat uit een conjunctie van de eigenschap E en een link met de inwendige naamwoordgroep via een onbekend predicaat. Het is x die aan eigenschap E moet voldoen omdat de entiteit van het substantief het belangrijkste is en de entiteit van de naamwoordgroep als geheel voorstelt.

Bij de tweede grammaticale regel is er geen expliciet lidwoord aanwezig maar de betekenis hiervan is wel nog steeds nodig om een scope te geven aan het zelfstandig naamwoord. Er wordt dus gebruik gemaakt van dezelfde betekenis alsof er wel een lidwoord had gestaan (lijn 10).

Beiden naamwoordgroepen krijgen de waarde `np` voor de feature `coord` naar de extra naamwoordgroep die zich in deze naamwoordgroep bevindt.

```
1  np([coord:np, num:Num, gap:[], sem:Sem]) -->
2    det([num:Num, sem:Det]),
3    np([coord:no, num:_, gap:[], sem:NP]),
4    n([num:Num, sem:N]),
5    { Sem = lam(E, app(app(Det, N), lam(X, merge(app(E, X), app(NP, lam(Y,
6      drs([], [rel(_, X, Y)])))))))) }.
```

```
7
8  np([coord:np, num:Num, gap:[], sem:Sem]) -->
9    { semLex(det, [num:sg, sem:Det]) },
10   np([coord:no, num:_, gap:[], sem:NP]),
11   [s],
12   n([coord:_, num:Num, sem:N]),
13   { Sem = lam(E, app(app(Det, N), lam(X, merge(app(E, X), app(NP, lam(Y,
14     drs([], [rel(_, X, Y)])))))) }.
```

Grammatica 6.3: De grammaticale regels voor een naamwoordgroep met een onbekende relatie

6.3.3 Een vergelijkende naamwoordgroep

Grammatica 6.4 geeft de grammaticale regels voor een vergelijkende naamwoordgroep, zoals "3 years after John". Zo'n naamwoordgroep bestaat uit een hoeveelheid (`quantity`), een comparatief (`comp`) en een andere naamwoordgroep (`np`). Deze laatste moet een simpelere naamwoordgroep zijn (namelijk een eenvoudige naamwoordgroep of één met een onbekende relatie). De betekenis bestaat uit twee simpele lambda-applicaties $[\![np]\!] = ([\![comp]\!]\,([\![np_1]\!]))\,([\![np_2]\!])$

Zo'n hoeveelheid kan onbepaald zijn zoals "sometime" (lijn 7-9) of een simpele naamwoordgroep zoals "2 years" (lijn 10-12).

Deze naamwoordgroep heeft de waarde `comp` voor de feature `coord`. Die feature is nodig om niet in een oneindige lus te geraken. Een vergelijkende naamwoordgroep kan namelijk starten met een simpele naamwoordgroep (lijn 2 en 10-12). Zonder de feature `coord` zou een vergelijkende naamwoordgroep ook kunnen starten met een (andere) vergelijkende naamwoordgroep. Bij het zoeken of de volgende woorden een vergelijkende naamwoordgroep kunnen zijn, gaat prolog eerst kijken of ze een vergelijkende naamwoordgroep zijn. Op die manier geraakt prolog in een lus. Bij het gebruik van een andere parser zou dit niet per se nodig zijn.

Binnen logigrammen zit er ook altijd een ellips in zo'n vergelijkende naamwoordgroep, namelijk die van het hoofdwerkwoord binnen de naamwoordgroep waarmee vergeleken wordt. In de zin "John finished after Mary" ontbreekt het woord "finished" op het einde van de zin: "John finished after Mary [finished]". De feature `gap` op lijn 1 en 4 zegt dat de hele naamwoordgroep een ellips heeft, met name in de naamwoordgroep waarmee vergeleken wordt. Binnen logigrammen is die ellips er altijd. De grammatica staat niet toe dat er geen ellips is.

Lijn 14 t.e.m. 20 geven de betekenis van zo'n naamwoordgroep met een ellips. In formulevorm wordt dit

$$[\![np]\!] = \lambda E \cdot \left\{ \boxed{\begin{array}{c} z \\ \hline \\ \end{array}} \oplus E\,(z) \oplus [\![tv]\!]\,(\lambda E2 \cdot E2\,(z))\,([\![np_1]\!]) \right\}$$

De naamwoordgroep voldoet aan de eigenschap E als er een z bestaat die hier aan voldoet en bovendien zodang is dat er voldaan is aan de betekenis van het werkwoord met die z als lijdend voorwerp[2] en np_1 als onderwerp.

[2] $\lambda E2 \cdot E2\,(z)$ kan men opnieuw zien als een eigennaam die verwijst naar een bepaalde z

```
1  np([coord:comp, num:Num, gap:[tv:TV | G], sem:Sem]) -->
2    quantity([num:Num, sem:NP1]),
3    comp([sem:Comp]),
4    { Coord = no; Coord = np },
5    np([coord:Coord, num:_, gap:[tv:TV | G], sem:NP2]),
6    { Sem = app(app(Comp, NP1), NP2) }.
7
8  quantitity([num:_, sem:S]) -->
9    some([sem:S]),
10   { Sem = S }.
11 quantity([num:Num, sem:Sem]) -->
12   np([coord:no, num:Num, gap:[], sem:NP]),
13   { Sem = NP}.
14
15 np([coord:no, num:Num, gap:[tv:TV | G], sem:Sem]) -->
16   np([coord:_, num:Num, gap:G, sem:NP]),
17   { Sem = lam(E, merge(
18       merge(
19         drs([Z], []),
20         app(E, Z)),
21       app(app(TV, lam(E2, app(E2, Z))), NP))) }.
```

Grammatica 6.4: De grammaticale regels i.v.m. een vergelijkende naamwoordgroep

6.3.4 Een naamwoordgroep met een voegwoord

Ten slotte is er nog een naamwoordgroep met een voegwoord in of ook wel gecoördineerde naamwoodgroep genoemd. Grammatica 6.5 geeft de twee grammaticale regels die dit soort naamwoordgroepen beschrijven. De eerste regel zegt dat een gecoördineerde naamwoordgroep kan bestaat uit een eventuele prefix van een voegwoord, een naamwoordgroep (een simpele, één met een onbekende relatie of een vergelijkende naamwoordgroep), een voegwoord en opnieuw een naamwoordgroep (een simpele, één met een onbekende relatie, één met een vergelijking of een naamwoordgroep met een voegwoord van het zelfde type als het voegwoord van deze naamwoordgroep).

De tweede grammaticale regel behandelt het geval dat het voegwoord is verzwegen, zoals in “John, Pete and Mary”. De naamwoordgroep “John, Pete” bevat geen voegwoord. De grammaticale regel is vrij gelijkaardig. De enige andere verandering is dat de tweede naamwoordgroep sowieso opnieuw gecoördineerd moet zijn. Het type van voegwoord in die tweede naamwoordgroep is ook het type van het voegwoord dat is verzwegen.

Het getal van een gecoördineerde naamwoordgroep is afhankelijk van het type van het voegwoord (lijn 2, 12 en 20-22). Een conjunctief voegwoord (“and”) resulteert in een naamwoordgroep in het meervoud. De andere twee types resulteren in een naamwoordgroep in het enkelvoud.

De betekenis bestaat zoals bij zo vele grammaticale regels uit twee lambda-applicaties.

$$[\![np]\!] = [\![coord]\!]\,([\![np_1]\!])\,([\![np_2]\!])$$

```
1  np([coord:CoordType, num:CoordNum, gap:G, sem:Sem]) -->
```

```
2   { npCoordNum(CoordType, CoordNum) },
3   coordPrefix([type:CoordType]),
4   { Coord1 = no ; Coord1 = np ; Coord1 = comp },
5   np([coord:Coord1, num:sg, gap:G, sem:NP1]),
6   coord([type:CoordType, sem:Coord]),
7   { Coord2 = CoordType ; Coord2 = no ; Coord2 = np },
8   np([coord:Coord2, num:_, gap:G, sem:NP2]),
9   { Sem = app(app(Coord, NP1), NP2) }.
10
11 np([coord:CoordType, num:CoordNum, gap:G, sem:NP, vType:Type]) -->
12   { npCoordNum(CoordType, CoordNum) },
13   coordPrefix([type:CoordType]),
14   { Coord1 = no ; Coord1 = np },
15   np([coord:Coord1, num:sg, gap:G, sem:NP1, vType:Type]),
16   noCoord([type:CoordType, sem:Coord]),
17   np([coord:CoordType, num:_, gap:G, sem:NP2, vType:Type]),
18   { Sem = app(app(Coord, NP1), NP2) }.
19
20 npCoordNum(conj, pl).
21 npCoordNum(disj, sg).
22 npCoordNum(neg, sg).
```

Grammatica 6.5: De grammaticale regels voor een naamwoordgroep met een voegwoord

6.4 Verbale constituent

6.4.1 Een simpele verbale constituent

Grammatica 6.6 geeft de grammaticale regels voor een aantal simpele verbale constituenten.

De eenvoudigste verbale constituent bestaat uit een transitief werkwoord, gevolgd door haar prefix, een naamwoordgroep (als lijdend voorwerp) en ten slotte het achtervoegsel van het werkwoord. Het getal van het werkwoord is ook deze van de verbale constituent. De betekenis is een lambda-applicatie $[\![vp]\!] = [\![tv]\!]([\![np]\!])$.

De tweede grammaticale regel drukt iets gelijkaardigs uit maar nu in het geval het lijdend voorwerp een vergelijking is (`coord:comp`) of een conjunctie van twee vergelijkingen (`coord:conj`). In dat geval zijn het voor- en achtervoegsel optioneel. Neem bijvoorbeeld de zinnen “John won in 2008” en “Mary won 2 years after John”. Het voorvoegsel “in” verdwijnt in het geval van een vergelijkende naamwoordgroep. Bovendien is er een ellips van het werkwoord in de naamwoordgroep (zie ook sectie 6.3.3). Dit reflecteert zich in de feature `gap` van de naamwoordgroep.

Merk op dat we in de eerste grammaticale regel niet toestaan dat er een ellips is van het werkwoord binnen het lijdende voorwerp. Dit zou kunnen lijden tot extra betekenissen die niet nuttig zijn binnen een logigram.

In bijzinnen kan het soms voorkomen dat er een naamwoordgroep voor het werkwoord komt te staan, bijvoorbeeld “The horse that Mary won, ...”. In dat geval is de naamwoordgroep van de verbale constituent het onderwerp en de naamwoordgroep waar de bijzin bijstaat het lijdend voorwerp. De derde grammaticale regel (lijn 14-17) drukt dit

uit. De semantiek is

$$[\![vp]\!] = \lambda L \cdot [\![tv]\!](L)([\![np]\!])$$

Een verbale constituent kan ook een hulpwerkwoord bevatten. In dat geval komt het getal en de vorm (infinitief, deelwoord, ...) van de constituent overeen met dat van het hulpwerkwoord. Het hoofdwerkwoord moet in zo'n constituent voorkomen als infinitief (`inf`) of als deelwoord (`part`). De semantiek is nogmaals een lambda-appliatie

$$[\![vp]\!] = [\![av]\!]([\![vp_1]\!])$$

```
1  vp([coord:no, inf:I, num:Num, gap:G, sem:Sem]) -->
2    tv([inf:I, num:Num, positions:Pre-Post, sem:TV]),
3    Pre,
4    np([coord:_, num:_, gap:G, sem:NP]),
5    Post,
6    { Sem = app(TV, NP) }.
7  vp([coord:no, inf:I, num:Num, gap:G, sem:Sem]) -->
8    tv([inf:I, num:Num, positions:Pre-Post, sem:TV]),
9    optional(Pre),
10   { Coord = conj ; Coord = comp },
11   np([coord:Coord, num:_, gap:[tv:TV | G], sem:NP]),
12   optional(Post),
13   { Sem = app(TV, NP) }.
14
15 vp([coord:no, inf:I, num:Num, gap:[], sem:Sem]) -->
16   np([coord:_, num:_, gap:[], sem:NP]),
17   tv([inf:I, num:Num, positions:_-[], sem:TV]),
18   { Sem = lam(L, app(app(TV, L), NP))}.
19
20 vp([coord:no, inf:Inf, num:Num, gap:[], sem:Sem]) -->
21   av([inf:Inf, num:Num, sem:AV]),
22   { Inf2 = inf ; Inf2 = part },
23   vp([coord:_, inf:Inf2, num:_, gap:[], sem:VP]),
24   { Sem = app(AV, VP) }.
25
26
27 optional(X) -->
28   X.
29 optional(X) -->
30   [].
```

Grammatica 6.6: De grammaticale regels voor een simpele verbale constituent

6.4.2 Een verbale constituent met koppelwerkwoord

Sectie 5.9 maakte een onderscheid tussen drie betekenissen van een koppelwerkwoord afhankelijk van de categorie van het *argument* van dat werkwoord. Sectie 6.1.3 beschreef reeds het geval waarbij een adjectiefzin het argument is. Grammatica 6.7 beschrijft de grammatica voor een koppelwerkwoord met een nominale of voorzetselconstituent. De eerste twee grammaticale regels van grammatica 6.7 zeggen namelijk dat een verbale constituent kan bestaan uit een koppelwerkwoord met een nominale constituent of met een

voorzetselconstituent. Het getal en de vorm van de verbale constituent komen overeen met die van het werkwoord. De betekenis is voor beide regels een simpele lambda-applicatie.

De derde grammaticale regel drukt een *alldifferent*-constraint uit. Deze is van toepassing voor zinnen als "John and Mary are two different persons" of "John, the person with the horse and the man from France are all different people". Deze verbale constituent bestaat uit een koppelwerkwoord; een getal of het woordje "all"; het woord "different"; en tenslotte en substantief. De betekenis is

$$[\![vp]\!] = \lambda N \cdot N \left(\lambda x \cdot \boxed{\begin{array}{|l} \\ \hline alldifferent(x) \end{array}} \right)$$

Dit drukt uit dat de entiteit van het onderwerp deel is van een alldifferent constraint. Zoals we in sectie 5.11 reeds aanhaalden is er geen collectieve lezing voor een naamwoordgroep. Maar een alldifferent constraint drukt uit dat er binnen het collectief van het onderwerp geen twee dezelfde entiteiten bevinden. We passen daarom de vertaling van DRS-structuren naar eerste-orde-logica lichtjes aan om dit te omzeilen.

$$\left(\begin{array}{|l} \hline x_1, \ldots, x_n \\ \hline \gamma_1 \\ \ldots \\ \gamma_m \\ alldifferent(y_1) \\ \ldots \\ alldifferent(y_k) \end{array} \right)^{fo} = \left(\begin{array}{|l} \hline x_1, \ldots, x_n \\ \hline \gamma_1 \\ \ldots \\ \gamma_m \\ y_1 \neq y_2 \\ \ldots \\ y_1 \neq y_k \\ alldifferent(y_2) \\ \ldots \\ alldifferent(y_k) \end{array} \right)^{fo}$$

en

$$\left(\begin{array}{|l} \hline x_1, \ldots, x_n \\ \hline \gamma_1 \\ \ldots \\ \gamma_m \\ alldifferent(y_k) \end{array} \right)^{fo} = \left(\begin{array}{|l} \hline x_1, \ldots, x_n \\ \hline \gamma_1 \\ \ldots \\ \gamma_m \end{array} \right)^{fo}$$

M.a.w. alle alldifferent constraints die zich binnen één DRS-structuur bevinden, behoren tot één groep van entiteiten die allemaal verschillend zijn. Men kan dit ook schalen naar meerdere groepen door de groep te reïficeren binnen de alldifferent constraint (d.w.z. $alldifferent(g_i, x_j)$ i.p.v. $alldifferent(x_j)$).

Er wordt geen rekening gehouden met de betekenis van het getal of het koppelwerkwoord. "John and Mary are three different persons." is dus een geldige zin, alhoewel de "three" niet overeenkomt met het aantal entiteiten in het onderwerp.

```
1  vp([coord:no, inf:Inf, num:Num, gap:[], sem:Sem]) -->
2    cop([type:np, inf:Inf, num:Num, sem:Cop]),
3    np([coord:_, num:_, gap:[], sem:NP]),
4    { Sem = app(Cop, NP) }.
5
6  vp([coord:no, inf:Inf, num:Num, gap:[], sem:Sem]) -->
7    cop([type:pp, inf:Inf, num:Num, sem:Cop]),
```

```
8   pp([sem:PP]),
9   { Sem = app(Cop, PP) }.
10
11  vp([coord:no, inf:Inf, num:Num, gap:[], sem:Sem]) -->
12    cop([type:np, inf:Inf, num:Num, sem:_]),
13    numberOrAll(),
14    [different],
15    n([coord:_, num:Num, sem:_]),
16    { Sem = lam(N, app(N, lam(X, drs([], [alldifferent(X)])))) }.
17
18  numberOrAll() -->
19    number([sem:_]).
20  numberOrAll() -->
21    [all].
```

Grammatica 6.7: De grammaticale regels voor een verbale constituent met een koppelwerkwoord

6.4.3 Een verbale constituent met voegwoord

Een gecoördineerde verbale constituent is gelijkaardig aan die van een nominale constituent. Er is echter één klein verschil in de betekenis ervan. Een simpele lambda-applicatie volstaat hier niet altijd. Indien het onderwerp een kwantor bevat, dan valt de verbale constituent namelijk onder de scope van deze kwantor. We willen dus ook dat de vertaling van het voegwoord binnen de scope van de kwantor valt. Als we $[\![vp]\!] = [\![coord]\!]\,([\![vp_1]\!])\,([\![vp_2]\!])$ als vertaling nemen, dan valt de vertaling van het voegwoord buiten de scope van het onderwerp. Neem bijvoorbeeld "No man breathes and is dead". De bovenstaande semantiek zou in eerste-orde-logica de vertaling $(\neg\exists m \cdot man(x) \wedge breathes(x)) \wedge (\neg\exists m \cdot man(x) \wedge dead(x))$ geven. De juiste vertaling is $\neg(\exists m \cdot man(x) \wedge breathes(x) \wedge dead(x))$. De juiste semantiek is

$$[\![vp]\!] = \lambda O \cdot O\,(\lambda x_o \cdot ([\![coord]\!]\,([\![vp_1]\!])\,([\![vp_2]\!]))\,(\lambda E \cdot E\,(x_o)))$$

De zin is waar als het onderwerp voldoet aan de eigenschap dat *een eigennaam die verwijst naar* x_o[3] het onderwerp vormt van de coördinatie van de twee onderliggende verbale constituenten. In geval van een conjunctie met het dus één x_o zijn die voldoet aan beide verbale constituenten. Met de eerste semantiek zou dit een verschillende x_o kunnen zijn.

Voor logigrammen leiden beide betekenissen echter tot een correct resultaat. Het onderwerp zal namelijk altijd volledig gespecifieerd zijn. Er is maar één entiteit die aan de omschrijving voldoet. Daardoor maakt het niet uit dat er twee kantoren zijn die gecombineerd worden met het voegwoord in kwestie i.p.v. dat het voegwoord zich bevindt in de scope van de kwantor. Dit voorbeeld toont echter nog eens aan dat het opstellen van de formules niet altijd triviaal is.

```
1   vp([coord:yes, inf:Inf, num:Num, gap:[], sem:Sem]) -->
2     vp([coord:no, inf:Inf, num:Num, gap:[], sem:VP1]),
3     coord([type:_, sem:Coord]),
4     vp([coord:_, inf:Inf, num:Num, gap:[], sem:VP2]),
```

[3] Een eigennaam die verwijst naar x_o heeft als semantiek $\lambda E \cdot E\,(x_o)$.

```
5   { CoordinatedVP = app(app(Coord, VP1), VP2) },
6   { Sem = lam(O, app(O, lam(X, app(CoordinatedVP, lam(E, app(E, X)))))) }.
```

Grammatica 6.8: De grammaticale regels voor een verbale constituent met een voegwoord

6.5 Zin

Er zijn drie soorten zinnen binnen logigrammen. Een normale zin bestaat uit een nominale en een verbale constituent die overeenkomen in getal. Dat is de eerste grammaticale regel van grammatica 6.9. De betekenis is een simpele lambda-applicatie $[\![s]\!] = [\![vp]\!]([\![np]\!])$.

Een tweede soort zin is een andere vorm voor een alldifferent constraint en wordt gebruikt om alle domeinelement van een bepaald domein op te sommen. Een voorbeeldzin is “The four players are John, Mary, the person who plays with cards and the man from France”. Zo’n zin bestaat uit het woordje “the”, een getal (het aantal domeinelementen), een substantief, een koppelwerkwoord en ten slotte een gecoördineerde naamwoordgroep. De betekenis is gelijkaardig aan die van de alldifferent-constraint uit sectie 6.4.2

Een laatste soort zin is van de vorm “Of John and Mary, one is from France and the other plays with cards”. De zin bestaat dus uit twee nominale en twee verbale constituenten. De twee entiteiten van de nominale constituenten zijn verschillend en bovendien is elk het onderwerp van één van de twee verbale constituenten.

$$[\![s]\!] = [\![np_1]\!]\left(\lambda x_1 \cdot [\![np_2]\!]\left(\lambda x_2 \cdot \left\{ \begin{array}{|c|}\hline \\ \hline x_1 \neq x_2 \\ \hline \end{array} \oplus \begin{array}{|l|}\hline \\ \hline [\{[\![vp_1]\!](\lambda E \cdot E(x_1)) \oplus [\![vp_2]\!](\lambda E \cdot E(x_2))\}] \\ \vee \\ [\{[\![vp_1]\!](\lambda E \cdot E(x_2)) \oplus [\![vp_2]\!](\lambda E \cdot E(x_1))\}] \\ \hline \end{array} \right\}\right)\right)$$

We gaan er van uit dat beide naamgroepen volledig zijn gespecifieerd. Er is dus maar één mogelijke x_1 en x_2. We kunnen x_1 dus ook voorstellen als een naamwoordgroep met semantiek $\lambda E \cdot E(x_1)$. Zo’n naamwoordgroep voldoet immers aan een eigenschap E als en slechts als x_1 dat doet. Een analoge redenering geldt voor x_2. Dan zegt de semantiek dat x_1 en x_2 verschillend moeten zijn en dat x_1 het onderwerp vormt voor de eerste verbale constituent en x_2 voor de tweede verbale constituent, of omgekeerd.

```
1   s([coord:no, sem:Sem]) -->
2     np([coord:_, num:Num, gap:[], sem:NP]),
3     vp([coord:_, inf:fin, num:Num, gap:[], sem:VP]),
4     { Sem = app(VP, NP) }.
5
6   s([coord:no, sem:Sem]) -->
7     [the],
8     number([sem:_, vType:_]),
9     n([coord:_, num:pl, sem:_]),
10    cop([type:np, inf:fin, num:pl, sem:_]),
11    np([coord:conj, num:_, gap:[], sem:NP]),
12    { Sem = app(NP, lam(X, drs([], [alldifferent(X)]))) }.
13
14  s([coord:no, sem:Sem]) -->
```

```
15  [of],
16  np([coord:_, num:sg, gap:[], sem:NP1]),
17  [and],
18  np([coord:_, num:sg, gap:[], sem:NP2]),
19  [one],
20  vp([coord:no, inf:fin, num:sg, gap:[], sem:VP1]),
21  [and, the, other],
22  vp([coord:no, inf:fin, num:sg, gap:[], sem:VP2]),
23  { Sem = app(NP1, lam(X1, app(NP2, lam(X2,
24      merge(
25        drs([], [not(drs([], [eq(X1, X2)]))]),
26        drs([], [or(
27          merge(
28            app(VP1, lam(E, app(E, X1))),
29            app(VP2, lam(E, app(E, X2)))),
30          merge(
31            app(VP1, lam(E, app(E, X2))),
32            app(VP2, lam(E, app(E, X1)))))]))))))) }.
```

Grammatica 6.9: De grammaticale regels voor een zin

6.6 Conclusie

Onze grammatica voor logigrammen is redelijk beperkt. De betekenis van een grammaticale regel komt vaak neer op een simpele lambda-applicatie maar soms zijn ook complexere formules nodig om de scope van de kwantoren correct te krijgen.

Sommige grammaticale regels zijn algemeen bruikbaar. Andere zijn specifiek voor logigrammen. Zo is er een *alldifferent*-constraint en zijn er vergelijkingen (bijvoorbeeld "John is 2 years younger than Mary") die zich vertalen in een som of een verschil van twee getallen.

De grammatica is opgesteld om maar één vertaling toe te staan. Zo laten we enkel binnen een vergelijkende naamwoordgroep een ellips van het werkwoord toe. In andere gevallen zou dit leiden tot een extra vertaling. Bovendien is dit de enige plaats waar er een ellips van het werkwoord voorkomt in de bestudeerde logigrammen. Als we deze grammatica willen uitbreiden naar andere specificaties, moeten we meerdere betekenissen toelaten of bijvoorbeeld het gebruik van ellipsen verbieden.

Hoofdstuk 7

Types

In dit hoofdstuk voegen we types toe aan het framework van Blackburn en Bos. Aan de hand hiervan kunnen we vertalen naar een getypeerde logica en kunnen we automatisch de domeinen van een logigram afleiden uit de zinnen in natuurlijke taal.

We beschrijven eerst het achterliggende idee en hoe we dit kunnen toepassen op logigrammen. Dan bekijken we welke aanpassingen het lexicon en de grammatica moeten ondergaan. Ten slotte leggen we uit hoe we de domeinen van een logigram automatisch kunnen afleiden.

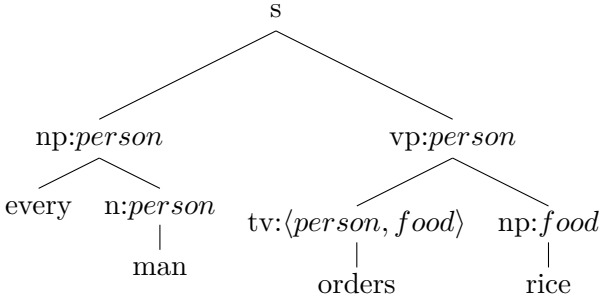
7.1 Het achterliggende idee

In natuurlijke taal kan men vele zinnen vormen zonder echte betekenis die wel grammaticaal correct zijn. Bijvoorbeeld de zin "Het gras drinkt het zingende huis" houdt weinig steek. Idealiter zouden we zo'n zinnen willen bestempelen als foutief. Gras is namelijk niet iets dat kan drinken en een huis kan ook niet zingen of gedronken worden. We zeggen dat ze slecht getypeerd zijn.

In de bestaande literatuur over gecontroleerde natuurlijke talen (CNL's) wordt hier weinig of geen aandacht aan besteed[1]. Deze thesis introduceert types in het framework van Blackburn en Bos.

Concreet voegen we types toe als extra feature aan de meeste grammaticale en lexicale categorieën. Net zoals de feature getal ervoor zorgde dat het onderwerp en het werkwoord van een zin overeenkomen in getal, zorgt de feature `vType`[2] dat het onderwerp en de verbale constituent overeenkomen in type.

Bijvoorbeeld voor "every man orders rice":



[1] In hoofdstuk 2 was RuleCNL de enige taal die ook gebruik maakt van types.

[2] We kiezen voor `vType` omdat sommige grammaticale categorieën reeds een feature `type` hebben. De `v` staat voor vocabularium.

Een naamwoordgroep krijgt een type dat overeenkomt met het type van de entiteit waarnaar het verwijst. Het type van een verbale constituent is gelijk aan dat van het onderwerp. Het type van een transitief werkwoord bestaat dan weer uit een type-paar. Het eerste voor het type van het onderwerp (en dus de verbale constituent) en het andere voor het type van het lijdende voorwerp. Ook een voorzetsel heeft een paar als waarde voor `vType`. Het ene komt overeen met dat van het substantief dat getransformeerd wordt en het andere komt overeen met dat van de naamwoordgroep uit de voorzetselconstituent.

Als men het lexicon annoteert met types, kan het systeem zinnen in natuurlijke taal type-checken. In deze thesis houden we het lexicon echter zo simpel mogelijk. De gebruiker moet daarom geen types opgeven. Het systeem zal aan type-inferentie doen en deze types proberen afleiden. Concreet willen we dus te weten komen dat "France" en "Italy" van hetzelfde type zijn, zonder per se te weten wat het type juist is. We hoeven dus niet te weten dat het om landen gaat. Wat ons vooral interesseert is het groeperen van de verschillende eigennamen volgens hun domein.

In sommige toepassingen van het KBS-paradigma, zoals requirements engineering, wordt er eerst een domein opgesteld. Dit domein zou dan ook in een lexicon met types gegoten kunnen worden. Daarna kan men een specificatie schrijven die ook juist getypeerd is. Zo kan men fouten in de specificatie of het domein opsporen. Men zou bijvoorbeeld niet kunnen spreken over het bestellen van drank als men in het domeinmodel enkel eten kan bestellen. Verder onderzoek over de haalbaarheid en nuttigheid hiervan is nodig.

7.2 Types voor logigrammen

Voor type-inferentie in logigrammen maken we één aanname: elk woord uit het lexicon heeft binnen één logigram exact één type. Later zal blijken dat hier niet altijd aan voldaan is (zie hoofdstuk 9). Het type-systeem zelf is vrij simpel: het bestaat uit een aantal basistypes en type-paren. Een type-paar moet altijd twee basistypes als argument hebben. Er is geen type-hiërarchie. Men kan dus niet uitdrukken dat mensen en dieren allebei levende wezens zijn. Type-checking is volledig gebaseerd op unificatie. Een zin is geldig als het type van het onderwerp en van de verbale constituent unificeren. We onderscheiden twee soorten basistypes: een basistype kan ofwel numeriek zijn ofwel niet.

Met een getypeerde natuurlijke taal kunnen we ook vertalen naar een getypeerde formele taal. Daarvoor breiden we DRS-structuren uit met types: elke entiteit is van een bepaald type. Vanaf nu vertalen we ook altijd naar eerste-orde-logica met types. De vertaling van een getypeerde DRS naar getypeerde eerste-orde logica is analoog met de vertaling van DRS naar eerste-orde-logica. De types worden vertaald zoals men verwacht:

$$\left(\begin{array}{|l} \hline x_1[t_1], \ldots, x_n[t_n] \\ \hline \gamma_1 \\ \ldots \\ \gamma_m \\ \hline \end{array}\right)^{fo} = \exists x_1[t_1] \ldots \exists x_n[t_n] \left((\gamma_1)^{fo} \wedge \ldots \wedge (\gamma_m)^{fo}\right)$$

$$\left(\begin{array}{|l} \hline x_1[t_1], ..., x_n[t_n] \\ \hline \gamma_1 \\ ... \\ \gamma_m \\ \hline \end{array} \Rightarrow B\right)^{fo} = \forall x_1[t_1] \ldots \forall x_n[t_n] \left(\left((\gamma_1)^{fo} \wedge ... \wedge (\gamma_m)^{fo}\right) \Rightarrow \left(B\right)^{fo}\right)$$

7.3 Aanpassingen

In het lexicon geven we elk woord één type dat nog onbekend is. De vertaling van een zelfstandig naamwoord is nu veel simpeler: $[\![noun]\!] = \lambda P \cdot \boxed{\;\;}$. Dit komt overeen met $[\![noun]\!] = \lambda P \cdot true$ in eerste-orde-logica. Er is geen restrictie meer vanuit het zelfstandig naamwoord, deze restrictie zit volledig in het type van de entiteit die we opleggen bij het quantificeren. Een determinator heeft namelijk als vertaling

$$[\![det]\!] = \lambda R \cdot \lambda S \cdot \left(\boxed{\begin{array}{|c|}\hline x[Type] \\ \hline \end{array}} \oplus R(x) \oplus S(x) \right)$$

Hierbij komt *Type* overeen met de waarde van de feature `vType`. Via unificatie krijgt de determinator het type van het zelfstandige naamwoord mee.

Ook bij een comparatief en een onbepaald woord is er een introductie van een nieuwe variabele. Bij een comparatief gebeurt dit gelijkaardig aan de determinator. Het type komt namelijk uit een lexicale feature `vType` die via de grammatica geünificeerd wordt met andere types.

Een onbepaald woord introduceert een positief geheel getal (zie sectie 5.10). De x heeft dus *int* als type.

$$[\![some]\!] = \lambda P \cdot \left\{ \boxed{\begin{array}{|c|}\hline x[int] \\ \hline x > 0 \\ \hline \end{array}} \oplus P(x) \right\}$$

In de praktijk blijkt het echter nodig om het domein te beperken tot een subset van de gehele getallen. Anders eindigt de inferentie met onderliggende tool IDP [10] niet. Deze subset wordt voorgesteld door *TypeDiff*, een numeriek type waarvan het domein is afgeleid van het domein van *Type* (de waarde van de feature `vType`). Stel bijvoorbeeld dat *Type* het domein `{2007, 2008, 2009, 2010, 2011}` heeft, dan heeft *TypeDiff* het domein `{1, -1, 2, -2, 3, -3, 4, -4}`. Een getal uit *TypeDiff* stelt namelijk een verschil voor van twee getallen uit *Type*.

De uitbreiding van de grammatica is vrij voor de hand liggend. Grammatica 7.1 geeft een aantal voorbeelden. De eerste grammaticale regel zegt dat het type van het onderwerp moet unificeren met dat van de verbale constituent. Het type van een verbale constituent komt dan weer overeen met het eerste type van het type-paar van het transitieve werkwoord. Het tweede type van dat paar komt overeen met het lijdend voorwerp.

```
1  s([coord:no, sem:Sem]) -->
2    np([coord:_, num:Num, gap:[], sem:NP, vType:Type]),
3    vp([coord:_, inf:fin, num:Num, gap:[], sem:VP, vType:Type]),
4    { Sem = app(VP, NP) }.
5
6  vp([coord:no, inf:I, num:Num, gap:G, sem:Sem, vType:TypeSubj]) -->
7    tv([inf:I, num:Num, positions:Pre-Post, sem:TV, vType:TypeSubj-TypeObj]),
8    Pre,
9    np([coord:_, num:_, gap:G, sem:NP, vType:TypeObj]),
10   Post,
11   { Sem = app(TV, NP) }.
```

Grammatica 7.1: Een aantal grammaticale regels met type

7.4 Type Inferentie

Om de types automatisch af te leiden veronderstellen we dat elk woord uit het lexicon één type heeft (maar dat dit type niet wordt meegegeven door de gebruiker). Vervolgens zoeken we de types van alle woorden. Tijdens het vertalen worden de types geünificeerd volgens het principe van één type per woord. Neem bijvoorbeeld de zinnen "John lives in France" en "The man with the dog lives in Italy". Op basis hiervan weten we dat "France" en "Italy" hetzelfde type hebben, beiden zijn namelijk het lijdende voorwerp van "lives in". Ook weten we dat "John" een "man" is. Ten slotte weten we dat "lives in" een relatie is tussen een man en iets van het type van "France" en "Italy", terwijl "with" een relatie is tussen een man en iets van het type van "dog".

Als een logigram veel synoniemen bevat, kan het systeem deze types niet automatisch af leiden. Stel bijvoorbeeld dat "Mary is from Spain" deel uitmaakt van het vorige voorbeeld. Dan kan het systeem niet afleiden dat "Spain" tot hetzelfde domein behoort als "France" en "Italy". Het systeem weet namelijk niet dat "from" en "lives in" dezelfde relatie uitdrukken. De gebruiker kan dit ook niet ingeven in het lexicon. In dat geval wordt één domein nog steeds voorgesteld door de combinatie van een aantal types. Om de onderliggende axioma's van een logigram correct te kunnen uitdrukken, moeten deze types overeenkomen met de domeinen.

Daarom stelt het systeem een aantal ja-nee-vragen aan de gebruiker. Hiervoor gebruikt het zoveel mogelijk taalkundige informatie. Er zijn 4 soorten vragen: "Zijn A en B dezelfde relatie (synoniemen)?", "Zijn A en B een omgekeerde relatie?", "Is A een mogelijk lijdend voorwerp van B?" en "Zijn A en B van hetzelfde type?". De eerste drie soorten vragen zijn puur taalkundig en zouden herschreven kunnen worden "Is ... een goed getypeerde zin?", bijvoorbeeld "Is 'John lives in the dog' een goed getypeerde zin?".

De laatste soort vraag is eigenlijk het doel van het hele type-systeem, namelijk achterhalen of twee domeinelementen van hetzelfde type zijn. Indien een bepaald domein in het logigram altijd wordt voorgesteld door een naamwoordgroep met een onbekende relatie (zie sectie 6.3.2) dan heeft men geen informatie over het type van deze domeinelementen en geen mogelijke relatie waar deze domeinelementen deel van kunnen maken. Er is dus geen mogelijke taalkundige vraag in dat geval.

7.5 Conclusie en verder onderzoek

Met behulp van types is het mogelijk om natuurlijke taal te type-checken. Zo kan men betekenisloze zinnen uitsluiten. Bovendien kan men met behulp van type-inferentie een deel van de wereldkennis afleiden. Zo kunnen we weten dat "France" en "Italy" gelijkaardige concepten uitdrukken, zonder te weten wat die concepten zijn. Dat men dit kan afleiden is zeker niet nieuw (Mikolov et al. [16] behaalden straffere resultaten via neurale netwerken). Het is wel een nieuwe manier om tot die resultaten te komen. Er zijn minder zinnen voor nodig, maar ze moet wel een bepaalde grammatica en type-systeem volgen.

In de bestaande literatuur over CNL's is er meestal geen of een beperkte ondersteuning voor types. De resultaten van deze thesis suggereren echter dat onderzoek hiernaar zeker de moeite waard kan zijn.

Een vraag die naar boven komt, is of men via type-checken van natuurlijke taal het aantal fouten in specificaties kan verlagen. Staat het toe om een betere suggestietool te maken die niet alleen rekening houdt met de grammatica maar ook met informatie omtrent types? De types kunnen namelijk het aantal suggesties beperken om zo de kwaliteit van de tool te verbeteren.

In deze thesis werd er gebruik gemaakt van een zeer simpel type-systeem. Ook een onderzoek naar een moeilijker systeem, bijvoorbeeld met een hiërarchie van types, lijkt nodig. Bovendien moet er ook onderzocht worden of de techniek toepasbaar is op andere problemen dan logigrammen.

Hoofdstuk 8

Een volledige specificatie

Hoofdstuk 4 introduceerde een framework voor semantische analyse. Hoofdstukken 5 en 6 beschreven hoe we dit framework konden gebruiken voor het vertalen van logigrammen naar logica. Hoofdstuk 7 voegde types toe aan dit framework. Op basis van deze vier hoofdstukken kunnen we nu de zinnen van een logigram omzetten naar zinnen in eerste-orde-logica met types.

In dit hoofdstuk beschrijven we de ontbrekende delen die nodig zijn om een volledige specificatie op te stellen van een logigram. Voor deze specificatie ontbreekt nog een formeel vocabularium en een aantal axioma's die inherent zijn aan logigrammen. Hierbij zullen we ook gebruik maken van type-informatie.

8.1 Een formeel vocabularium

We stellen een formeel vocabularium op dat ook getypeerd is. De types worden vertaald naar een subset van de gehele getallen of naar constructed types. Constructed types zijn types die bestaat uit een set van constanten met twee extra axioma's ingebouwd, namelijk het Domain Closure Axiom en het Unique Names Axiom. Onder het Domain Closure Axiom verstaan we dat de set van constanten de enige mogelijke elementen zijn van dat type. Het Unique Names Axiom drukt dan weer uit dat alle constanten verschillend zijn van elkaar.

Als er een substantief is van een bepaald type, dan geeft die een naam aan het type. De resterende types worden gewoon genummerd (`type1`, `type2`, ...).

De meest voorkomende types zijn types die een niet-numeriek domein voorstellen. In dat geval zijn er eigennamen van dat type. De eigennamen vormen de domeinelementen en kunnen vertaald worden naar de constanten van de constructed type. Het is echter ook mogelijk dat er 1 domeinelement ontbreekt. Niet alle domeinelementen hoeven namelijk voor te komen in de zinnen van een logigram. De bekendste logigram is genoemd naar zo'n ontbrekend element. *De Zebrapuzzel* [27] is een logigram waar de lezer gevraagd wordt welke persoon een zebra heeft. De zebra komt voor de rest niet voor in het logigram.

De gebruiker moet daarom opgeven hoeveel elementen er in elk domein zitten. Het ontbrekende element wordt dan toegevoegd als een extra constante. Omwille van het Domain Closure Axiom mag deze constante zeker niet ontbreken. Bij de oplossing van het logigram kan er dan bijvoorbeeld `the_other_person` komen te staan. Men weet dan over welke persoon het gaat. Er kan immers maar één element ontbreken. Als er twee elementen zouden ontbreken, dan zou men het onderscheid niet kunnen maken tussen de twee elementen. Er is echter altijd een unieke oplossing.

Daarnaast is het mogelijk dat een niet-numerieke type niet overeenkomt met een domein van het logigram. Er zijn dan geen eigennamen van dat type. Het is een tussenliggend type om twee andere domeinen te verbinden. Bijvoorbeeld "tour" in de zin "John follows the tour with 54 people". Er zijn twee echte domeinen in deze zin: de personen (met hun naam) en de groottes van een groep.

Voor zo'n type, introduceren we een constructed type met evenveel constanten als er domeinelementen zijn. Deze constanten spelen geen rol buiten voor het verbinden van de andere types. We zullen dus ook symmetrie-brekende axioma's moeten opleggen aan zo'n types om tot een unieke oplossing te komen.

Ten slotte kan een type een numeriek domein van het logigram voorstellen. In tegenstelling tot voor een niet-numeriek type kunnen we hier de nodige domeinelementen niet afleiden uit de tekst. We hebben bovendien de exacte subset nodig om tot een unieke oplossing te komen. Het volstaat niet om alle getallen buiten één te kennen. Bovendien kan het zelfs zijn dat er meer dan één getal ontbreekt. Voor numerieke types vragen we dus de domeinelementen aan de gebruiker.

Het is mogelijk dat zo'n numeriek type nog een afgeleid type heeft. Dit gaat dan om een *TypeDiff* uit sectie 7.3. Het domein van het afgeleide type kan automatisch berekend worden uit het domein van het type dat een domein voorstelt.

De predicaten uit het formeel vocabularium worden allemaal geïnduceerd door voorzetsels en transitieve werkwoorden. De types van het predicaat worden bepaald door het type van het woord dat het induceerde. "lives in" is bijvoorbeeld een werkwoord dat een mens als onderwerp neemt en een land als lijdend voorwerp. `lives_in` zal daarom een predicaat zijn met een eerste argument van type `human` en tweede argument van type `country`.

Via types is het bovendien mogelijk om de ontbrekende predicaten uit sectie 6.3.2 (Een naamwoordgroep met onbekende relatie) te achterhalen. We weten het type van x en y en dus het domein. Bovendien is er binnen logigrammen altijd maar één functie tussen twee domeinen, namelijk de bijectie die we zoeken. We zoeken dus een werkwoord of voorzetsel met het juiste type-paar. Neem bijvoorbeeld "the 2008 graduate...". Aangezien "2008" een jaartal is, "graduate" een persoon en "graduated in" een werkwoord tussen een persoon en een jaartal kan de naamwoordgroep herschreven worden als "the graduate who graduated in 2008, ...". Indien er (minstens) één bestaat kunnen we het predicaat dat overeenkomt met dat woord kiezen. Dit verhoogt de leesbaarheid van de formele vertaling. Het kan echter zijn dat er zo geen woord bestaat. In dat geval introduceren we een nieuw predicaat met de juiste types.

We gebruiken voor alle predicaten de predicaat-syntax (en dus niet de functionele syntax). De theorie bevat de nodige axioma's om deze predicaten te beperken tot bijecties.

8.2 De ontbrekende axioma's

Naast de zinnen van een logigrammen zijn er ook altijd een aantal beperkingen die eigen zijn aan een logigram en die niet expliciet vermeld worden. Daarom bevat een correcte theorie voor logigrammen nog extra axioma's. Er zijn 5 soorten axioma's. Een deel hiervan is gebaseerd op de types van de predicaten.

- Elk predicaat is een bijectie. Voor elk predicaat is er dus een extra zin in de vorm van $\forall x \cdot \exists y \cdot pred(x, y) \wedge \forall y \cdot \exists x \cdot pred(x, y)$ die dit uitdrukken. Bijvoorbeeld voor

lives_in:

$$\forall x \cdot \exists y \cdot lives\_in(x, y) \wedge \forall y \cdot \exists x \cdot lives\_in(x, y).$$

- Er zijn 3 soorten axioma's die uitdrukken dat er equivalentieklassen bestaan van verschillende domeinelementen die samenhoren. Elke klasse bevat één domeinelement van elk domein. Twee elementen zijn equivalent als ze gelinkt zijn via een predicaat of als ze aan elkaar gelijk zijn (op die manier is aan reflexiviteit voldaan).
  - **Synonymie** Er is exact één bijectie tussen twee domeinen. Twee predicaten met dezelfde types zijn dus altijd synoniemen. $\forall x \forall y \cdot pred_1(x, y) \Leftrightarrow pred_2(x, y)$. M.a.w. er is maar één equivalentierelatie. Bijvoorbeeld voor $from(person, country)$ and $lives\_in(person, country)$:

    $$\forall x \forall y \cdot lives\_in(x, y) \Leftrightarrow from(x, y)$$

  - **Symmetrie** Predicaten met een omgekeerde signatuur stellen elkaars inverse voor $\forall x \forall y \cdot pred_1(x, y) \Leftrightarrow pred_2(y, x)$. Bijvoorbeeld $gave(student, presentation)$ en $given\_by(presentation, student)$:

    $$\forall x \forall y \cdot gave(x, y) \Leftrightarrow given\_by(y, x)$$

  - **Transitiviteit** Ten slotte zijn er nog axioma's om de verschillende bijecties te linken. Deze axioma's zorgen voor de transitiviteit van de equivalentierelatie. Bijvoorbeeld de predicaten $pred_1(t_x, t_y)$, $pred_2(t_x, t_z)$ en $pred_3(t_z, t_y)$ introduceren het axioma $\forall x \forall y \cdot pred_1(x, y) \Leftrightarrow \exists z \cdot pred_2(x, z) \wedge pred_3(z, y)$. Bijvoorbeeld voor $spoke\_for(student, time)$, $gave(student, presentation)$ en $lasted\_for(presentation, time)$:

    $$\forall x \forall y \cdot spoke\_for(x, y) \Leftrightarrow \exists z \cdot gave(x, z) \wedge lasted\_for(z, y).$$

- Niet-numerieke types die geen domein voorstellen introduceren een aantal nieuwe constanten (bijvoorbeeld het type van "tour" in "John follows the tour with 54 people"). Deze constanten zijn inwisselbaar. Om een unieke oplossing te bekomen introduceren we een symmetriebrekend axioma dat de constanten van zo'n type linkt aan de domeinelementen van een domein naar keuze. Bijvoorbeeld

  $$with(TourA, 54) \wedge with(TourB, 64) \wedge with(TourC, 74)$$

8.3 Conclusie

Een volledige specificatie van een logigram bestaat niet alleen uit de vertaling van de zinnen naar logica. Er is ook nood aan een formeel vocabularium en een aantal extra axioma's die niet expliciet vermeld worden. Beiden kunnen echter opgesteld worden aan de hand van informatie i.v.m. de types van woorden.

Zowel de constructie van het vocabularium als de extra axioma's zijn specifiek voor logigrammen. Zo is het domein van numerieke types bij logigrammen altijd beperkt tot een subset van de gehele getallen. Voor andere specificaties kan zo'n type vertaald worden naar de gehele getallen. Het is ook triviaal om in te zien dat de axioma's specifiek zijn voor logigrammen.

Hoofdstuk 9

Evaluatie

In dit hoofdstuk beschrijven we een experiment ter evaluatie van het framework. We controleren of de grammatica uit hoofdstuk 6 (opgesteld op basis van de eerste tien puzzels uit [20]) gebruikt kan worden om nieuwe logigrammen te parsen. Anderzijds is de vraag of de type-inferentie beschreven in hoofdstuk 7 genoeg is om logigrammen op te lossen. Kunnen we op basis van het principe van één type per woord en een aantal taalkundige ja-nee-vragen de juiste types achterhalen?

9.1 Experiment

Het experiment bestaat eruit om een grammatica op te stellen op basis van de eerste tien puzzels uit Puzzle Baron's Logic Puzzles Volume 3 [20]. Het resultaat is te vinden in hoofdstuk 6. Vervolgens passen we dit toe op de volgende tien puzzels uit hetzelfde boekje. Met dit experiment willen we een aantal vragen beantwoorden:

1. Is de grammatica van een logigram beperkt (binnen hetzelfde boekje, met name Puzzle Baron's Logic Puzzles Volume 3 [20])? M.a.w. moeten er aanpassingen gebeuren aan de puzzels in de testset om gevat te kunnen worden door de grammatica? Zo ja, hoeveel aanpassingen zijn er nodig en wat voor aanpassingen? Zijn het eerder kleine herschrijvingen of moeten de zinnen volledig anders geformuleerd worden? Is het überhaupt mogelijk om elk logigram te herschrijven?
2. Kunnen we op basis van het principe van één type per woord en een aantal taalkundige ja-nee-vragen de juiste types achterhalen?
3. Werkt het framework zoals beschreven in deze thesis voor het oplossen van logigrammen? Hoeveel logigrammen kunnen we oplossen? En indien we ze niet kunnen oplossen, waarom niet?

Een belangrijke opmerking bij de eerste vraag is dat de grammatica opgesteld wordt op basis van 10 logigrammen. Daardoor is de grammatica niet per se makkelijk te leren. De grammatica is namelijk zo complex als de eerste 10 puzzels samen. Bovendien is het niet per se erg als er aanpassingen moeten gebeuren aan de zinnen uit de testset. Dit wil vooral zeggen dat het boekje geen gestructureerde grammatica volgt. De belangrijkste vraag is vooral of het mogelijk is om de zinnen van elk logigram te herschrijven zodanig dat het systeem de puzzel kan omzetten naar een volledige en correcte specificatie. M.a.w. kan het systeem elke puzzel oplossen (nadat we eventueel de zinnen herschreven hebben)?

9.2 Trainingset

De trainingset bestaat uit de tien eerste puzzels uit Puzzle Baron's Logic Puzzles Volume 3 [20]. Bij het opstellen van de grammatica proberen we de puzzels zo weinig mogelijk aan te passen. Dit is echter niet altijd mogelijk. We bespreken de aanpassingen die we doen aan de trainingset en de reden hiervoor.

9.2.1 Logigram 4

Het eerste logigram dat aangepast wordt, is logigram 4. In drie van de tien zinnen staan overbodige tijdsaanduidingen:

- Of Lonnie and the person from Frenchboro, one is now with the Dodgers and the other graduated in 2005
- Lonnie currently plays for the Mariners
- The five players are the person from Frenchboro; Ivan; and the three players currently with the Indians Mariners and Giants

De laatste zin bevat bovendien nog een moeilijke opsomming om te verwerken. Hier komt namelijk een distributieve lezing van "the three players" aan te pas. Deze zin is daarom herschreven naar "The five players are the person from Frenchboro, Ivan, the player with the Indians, the person with the Mariners and the graduate that plays for the Giants"

9.2.2 Logigram 6

Ook logigram 6 bevat zo'n distributieve lezing. Bovendien is het deze keer in een "Of ... and ..., one ... and the other ..." constructie. Namelijk "Of the two dogs who graduated in March and April, one went to Tanager County and the other was assigned to Officer Ingram". Ook deze zin is herschreven. In dit geval naar "Of the dog who graduated in March and the dog who graduated in April, one went to Tanager County and the other was assigned to Officer Ingram"

9.2.3 Logigram 7

Logigram 7 heeft een vrij andere grammatica dan alle andere logigrammen. Bovendien overtreedt het een aantal veronderstellingen die we gebruiken voor het vertalen van de zinnen naar logica. Het logigram ziet er uit als volgt:

1. The Norwegian's birthday is May 18
2. Of Bill and the traveler born on June 14, one is from Norway and the other is from Canada
3. Izzy's birthday is 1 month after the politician's
4. The engineer is from France
5. The South African's birthday is in either April or May
6. The musician's birthday isn't in April
7. Jeffrey is either French or South African
8. The Canadian's birthday is 1 month after the surgeon's
9. Harry's birthday is sometime before Bill's

Zo gebruikt het logigram meerdere *eigennamen*[1] voor hetzelfde domeinelement (bv. “France” en “French”). Bovendien zijn er 3 domeinen die door elkaar gebruikt kunnen worden, met name de nationaliteit, de naam en het beroep van een persoon. Zo verwijst “Bill”, “the Canadian” en “the politician” allemaal naar dezelfde persoon. Ten slotte zijn er niet veel werkwoorden of voorzetsels. Dit maakt het vrij moeilijk om automatisch types af te leiden zonder aan de gebruiker te vragen of twee woorden van hetzelfde type zijn.

Het logigram kan wel herschreven worden om binnen het framework te passen. De zinnen liggen echter soms vrij ver af van het origineel. De nationaliteit wordt altijd aangegeven met de naam van het land. Bovendien introduceren we een werkwoord “acting as” als een soort van *type cast* tussen een beroep en een persoon. Ten slotte wordt het koppelwerkwoord vervangen door een echt werkwoord.

1. The traveler from Norway is born in May
2. Of Bill and the traveler born in June, one is from Norway and the other is from Canada
3. Izzy is born 1 month after the traveler acting as politician
4. The traveler acting as engineer is from France
5. The traveler from South Africa is born in either April or May
6. The musician’s birthday isn’t in April
7. Jeffrey is from either France or South Africa
8. The traveler from Canada is born 1 month after the traveler acting as surgeon
9. Harry is born sometime before Bill

9.2.4 Logigram 8

Logigram 8 is simpeler. Er zijn drie soorten aanpassingen die we doen.

1. We vervangen een adjectief door een voorzetselconstituent: “... the other was orange” wordt “... the other was in orange”.
2. In de twee zinnen met een vergelijking is de woordvolgorde een beetje anders. “The orange item required 5 fewer minutes to print than the yellow design” wordt “The orange item required 5 minutes fewer than the yellow design to print”
3. “just” wordt verwijderd in “Of the whistle and the blue piece, one took 30 minutes to print and the other took just 10”

9.2.5 Logigram 9

Logigram 9 bevat een aantal bijzinnen tussen haakjes. Deze worden gepromoveerd tot hoofdzinnen. Bijvoorbeeld “Opal (who isn’t 20 years old) is scheduled 2 hours after Harold” wordt “Opal isn’t 20 years old and is scheduled 2 hours after Harold”.

Daarnaast bevat logigram 9 een superlatief, namelijk “Elmer isn’t the youngest student”. Dit zou een nieuwe lexicale categorie kunnen zijn. Het is echter moeilijk om hier ook het juiste type van af te leiden. Er is gekozen om superlatieven te verbieden omdat binnen logigrammen dit altijd herschreven kan worden naar de juiste waarde. Deze zin wordt dan “Elmer isn’t 15 years old”.

[1] Merk op dat in we in deze thesis los omspringen met de term *eigennaam*. Elk domeinelement van een logigram moet voorgesteld worden met een eigennaam. Zie ook sectie 5.3.

9.2.6 Logigram 10

Logigram 10 heeft twee problemen. Enerzijds wordt er gebruik gemaakt van kommagetallen. Anderzijds is er niet voldaan aan het principe van één type per woord.

Aangezien de onderliggende tool IDP [10] geen kommagetallen aankan, ondersteunen we deze ook niet in de grammatica. We passen de zinnen aan. We tellen daarvoor bij de kommagetallen (bv. "$6.99") 0,01 op (tot bv. "$7"). Dit maakt geen verschil voor het logigram. Er wordt enkel gekeken naar de verschillen tussen twee getallen, die blijven met deze truk dezelfde.

Het werkwoord "to order" wordt in deze logigram gebruikt voor zowel het bestellen van pasta als de saus die erbij hoort. Daarom passen we één voorkomen van "to order" aan naar "to choose", zodanig dat "to order" enkel voor het bestellen van de pastasoort is en "to choose" enkel voor het kiezen van de bijhorende saus.

9.3 Testset

De testset bestaat uit de volgende tien logigrammen uit Puzzle Baron's Logic Puzzles Volume 3 [20]. Geen enkele logigram is vertaalbaar zonder wijzigingen uit te voeren aan de zinnen van het logigram. We bespreken de verschillende logigrammen en wat er aangepast moet worden om te vallen onder de grammatica van de trainingset. Daarbij minimaliseren we het aantal beperking en hoe groot deze moeten zijn.

9.3.1 Logigram 11: 1 aanpassing

Tabel 9.1 geeft een overzicht van de aanpassing aan logigram 11. Één zin werd herschreven van de passieve vorm naar de actieve vorm. Voor de rest past het logigram volledig binnen de grammatica van de testset.

Het type-systeem kan het domein van de sectoren van de aandelen wel niet achterhalen zonder op een vraag van de vierde soort terug te vallen. Namelijk een vraag of twee elementen tot hetzelfde domein behoren. Alle sectoren komen namelijk voor in naamwoordgroepen met een onbekende relatie ("the health-care sector", "the energy stock", ...).

Probleem	Origineel	Aangepast	Aantal
Passieve zin	The financial stock wasn't purchased by Edith	Edith didn't purchase the financial stock	1

Tabel 9.1: Een overzicht van de aanpassing aan logigram 11

9.3.2 Logigram 12: 5 aanpassingen

Tabel 9.2 geeft een overzicht van de aanpassingen aan logigram 12. In drie zinnen kwamen kommagetallen voor i.p.v. gehele getallen, net zoals in logigram 10 uit de trainingset. Dit is niet echt een probleem met de grammatica maar eerder met de onderliggende tool.

Daarnaast vervangen we een werkwoord door een ander werkwoord uit het logigram omdat aan het principe van één type per (werk)woord anders niet is voldaan.

Ten slotte passen we "the one" aan naar een naamwoordgroep met een echt substantief. "The one" is een grammaticale structuur waarbij "one" verwijst naar een substantief dat eerder in de zin komt. Dit kwam niet voor in de trainingset. Dit probleem had ook

opgelost kunnen worden door “one” als substantief toe te voegen aan het lexicon. Dit zou echter geen correcte evaluatie zijn van de opgestelde grammatica.

Probleem	Origineel	Aangepast	Aantal
Kommagetallen	Homer paid **$6.99**	Homer paid **$7**	3
Werkwoord heeft fout type	Glen paid $3 less than whoever **had** the sloppy joe	Glen paid $3 less than whoever **ordered** the sloppy joe	1
“the one”	The order with lemonade cost $1 more than **the one** with the water	The order with the lemonade cost $1 more than **the order** with the water	1

Tabel 9.2: Een overzicht van de aanpassingen aan logigram 12

9.3.3 Logigram 13: 3 aanpasssingen

Tabel 9.3 geeft een overzicht van de aanpassingen aan logigram 13. Er is opnieuw één zin met “the one”. Daarnaast is er nog één zin die twee fouten bevat. Enerzijds is er een constructie waarbij het voegwoord op een nieuwe plaats voorkomt. Anderzijds wordt er het koppelwerkwoord op een nieuwe manier gebruikt. Er zijn twee manieren om de zin te herschrijven. Bij de manier die aangegeven is in de tabel kan het type-systeem alle types achterhalen zonder te moeten terugvallen op een vraag van de vierde soort. Namelijk een vraag die stelt of twee domeinelementen tot hetzelfde domein behoren. De aanpassing die dichter bij het origineel licht is “...one is Hermans’s puppet...”.

Probleem	Origineel	Aangepast	Aantal
“the one”	The puppet going to Ypsilanti cost $250 more than **the one** going to St. Moritz	The puppet going to Ypsilanti cost $250 more than **the puppet** going to St. Moritz	1
Foute structuur naamwoordgroep	Of **the $1000 and $1250 dummies**, one is Herman’s and the other is going to Mexico city	Of **the $1000 dummy and the $1250 dummy**, one is Herman’s and the other is going to Mexico city	1
Fout gebruik koppelwerkwoord	Of the $1000 dummy and the $1250 dummy, one **is Herman’s** and the other is going to Mexico city	Of the $1000 dummy and the $1250 dummy, one **is from Herman** and the other is going to Mexico city	1

Tabel 9.3: Een overzicht van de aanpassingen aan logigram 13

9.3.4 Logigram 14: 13 aanpassingen

Tabel 9.4 geeft een overzicht van de aanpassingen aan logigram 14. Er zinnen twee zinnen met “the one”. Daarnaast is er een domeinelement dat twee keer voorkomt maar in verschillende vormen. Één keer als “monk’s fin shell” en één keer als “monk’s fin”. We passen één van de twee aan om dezelfde structuur te hebben.

In deze logigram zijn de kleuren adjectieven (zonder adjectiefconstituent). Dit wordt niet ondersteund in de grammatica. We beschouwen de adjectieven daarom als eigennamen en voegen telkens een voorzetsel toe om er een voorzetselconstituent van te maken.

Ten slotte wordt het werkwoord "found in" zowel gebruikt voor het jaartal als de plaats waar een schelp gevonden is. Dit is in strijd met het principe van één type per (werk)woord. We gebruiken daarom het werkwoord "discovered in" voor een plaats en "found in" voor een jaartal. In totaal zijn dit 5 aanpassingen.

Probleem	**Origineel**	**Aangepast**	**Aantal**
"the one"	The baby's ear shell was found 3 years before **the one** from Jamaica	The baby's ear shell was found 3 years before **the shell** from Jamaica	2
Dubbele vorm domeinelement	The shell found in 2001 is either the monk's fin or the coquina	The shell found in 2001 is either the monk's fin **shell** or the coquina	1
Fout gebruik koppelwerkwoord	The seashell found in Puerto Rico **isn't black and white**	The seashell found in Puerto Rico **isn't in black and white**	5
Werkwoord heeft fout type	The seashell **found in** Puerto Rico isn't in black and white	The seashell **discovered in** Puerto Rico isn't in black and white	5

Tabel 9.4: Een overzicht van de aanpassingen aan logigram 14

9.3.5 Logigram 15: 8 aanpassingen

Tabel 9.5 geeft een overzicht van de aanpassingen aan logigram 15. Andermaal bevat deze logigram een zin met "the one". Daarnaast is er een zin die een verwijzing maakt naar een vorige zin via het woord "also". Dit past niet in de grammatica. Bovendien is er net als in logigram 6 een constructie "Of the two ...". Deze constructie wordt op een gelijkaardige manier aangepast. Net als logigram 9, bevat deze logigram een superlatief. Ook hier wordt de superlatief vervangen door een eigennaam.

Ten slotte is er opnieuw een probleem met een werkwoord dat meer dan één type heeft. "got" wordt zowel gebruik voor de score van een student als die van een presentatie uit te drukken. Dit wordt aangepast naar "got" (voor studenten) en "received" (voor presentaties).

9.3.6 Logigram 16: 3 aanpassingen

Tabel 9.6 geeft een overzicht van de aanpassingen aan logigram 16. Enerzijds zijn er twee voorkomens van "the one" anderzijds is er een overtollig woord "always" dat verwijderd wordt.

9.3.7 Logigram 17: 6 aanpassingen

Tabel 9.7 geeft een overzicht van de aanpassingen aan logigram 17. Nogmaals zijn er drie zinnen met "the one". Daarnaast ontbreekt er in twee zinnen het woordje "trip". Ten slotte wordt "start at" meestal gebruikt voor een plaats en "begin at" voor een tijdstip. Er is echter één plaats waar dit niet zo was.

Probleem	Origineel	Aangepast	Aantal
"the one"	The presentation that got the A was 4 minutes shorter than **the one** on Caligula	The presentation that got the A was 4 minutes shorter than **the presentation** on Caligula	2
Overtollig woord	Yolada **also** didn't give a presentation on Galerius	Yolada didn't give a presentation on Galerius	1
"Of the two ..."	Of **the two presentations on Augustus and Caligula**, one was given by Catherine and the other lasted for 10 minutes	Of **the presentation on Augustus and the presentation on Caligula**, one was given by Catherine and the other lasted for 10 minutes	1
Superlatief	The talk on Nero was 2 minutes shorter than the presentation that got **the lowest grade**	The talk on Nero was 2 minutes shorter than the presentation that got **the D**	1
Werkwoord heeft fout type	The presentation that **got** the A was 4 minutes shorter than the one on Caligula	The presentation that **received** the A was 4 minutes shorter than the one on Caligula	3

Tabel 9.5: Een overzicht van de aanpassingen aan logigram 15

Probleem	Origineel	Aangepast	Aantal
"the one"	Floyd was either the juggler who went second or **the one** from Quasqueton	Floyd was either the juggler who went second or **the juggler** from Quasqueton	2
Overtollig woord	Floyd **always** juggles rubber balls	Floyd juggles rubber balls	1

Tabel 9.6: Een overzicht van de aanpassingen aan logigram 16

9.3.8 Logigram 18: 7 aanpassingen

Tabel 9.8 geeft een overzicht van de aanpassingen aan logigram 18. Grammaticaal is deze volledig correct. Er zijn echter veel problemen met types. Enerzijds wordt "finished with" en "received" zowel gebruikt met het beroep als onderwerp als met de persoon als onderwerp. Anderzijds wordt een beroep soms gelijk gesteld aan een persoon. We passen de zinnen aan zodat ze voldoen aan één type per woord. Hiervoor moeten we ook een soort van type-cast introduceren van beroep naar persoon ("the person acting as ...")

9.3.9 Logigram 19: 13 aanpassingen

Tabel 9.9 geeft een overzicht van de aanpassingen aan logigram 19. Er zijn drie zinnen met "the one". Daarnaast is er een zin met een bezittelijk voornaamwoord. Dit is een lexicale categorie die geen deel uitmaakt van de grammatica. De zin moet dus herschreven worden. In één zin is er een ellips van "comet".

In 6 zinnen is er een bijzin zoals "Whitaker discovered". Het probleem is dat het

Probleem	Origineel	Aangepast	Aantal
"the one"	Zachary's outing will begin 1 hour before **the one** starting at Casa Loma	Zachary's outing will begin 1 hour before **the outing** starting at Casa Loma	3
Ontbrekend woord (ellips)	Zachary's trip will begin 3 hours before Janice's	Zachary's trip will begin 3 hours before Janice's **trip**	2
Werkwoord heeft fout type	The Yorkville tour, the tour **starting at** 9 am and the one with 5 people are three different tours	The Yorkville tour, the tour **beginning at** 9 am and the one with 5 people are three different tours	1

Tabel 9.7: Een overzicht van de aanpassingen aan logigram 17

Probleem	Origineel	Aangepast	Aantal
Werkwoord heeft fout type	The academic **finished** 500 votes behind the teacher	The academic **received** 500 votes less than the teacher	4
Naamwoordgroep heeft fout type	Kelly Kirby finished 1000 votes ahead of **the academic**	Kelly Kirby finished 1000 votes ahead of **the person who acts as the academic**	3

Tabel 9.8: Een overzicht van de aanpassingen aan logigram 18

onduidelijk is of het gaat om de persoon die de planeet heeft ontdekt ("discovered by") of het jaartal waarin de planeet is ontdekt ("discovered in"). Indien de grammatica het passief correct zou ondersteunen of indien er type checking zou zijn i.p.v. type inferentie zou dit geen problemen geven. Nu zijn er echter twee betekenissen mogelijk. Daarom worden de zinnen herschreven.

Daarnaast wordt "year" zowel gebruikt voor de rotatielengte van een planeet als voor het aantal jaar tussen een ontdekking. Het principe van één type per woord is dus gebroken. Daarom wordt er "cycle" gebruikt tussen de ontdekkingen van twee planeten[2].

9.3.10 Logigram 20: 6 aanpassingen

Tabel 9.10 geeft een overzicht van de aanpassingen aan logigram 20. Ook logigram 20 bevat een voorkomen van "the one". Daarnaast zijn er 5 woordgroepen die verwijderd worden. Twee keer "said to be" en drie keer "Priscilla". Het is namelijk altijd Priscilla die een huis bezoekt.

9.3.11 Resultaten

Tabel 9.11 geeft een overzicht van alle aanpassingen. Er zijn een vijftal soort fouten. De eerste groep bestaat uit echte grammaticale elementen die niet ondersteund zijn. Dit zijn 30 van de 65 fouten (46%). De helft daarvan is opgelost door het ondersteunen van "the one" als naamwoordgroep. De tweede groep bestaat uit grammaticale elementen die

[2]Het is ook mogelijk om het andere gebruik van "year" te veranderen door een nieuw woord maar dit resulteert in minder aanpassingen.

Probleem	Origineel	Aangepast	Aantal
"the one"	The comet Tillman discovered, **the one** discovered in 2011 and Casputi are three different comets	The comet Tillman discovered, **the comet** discovered in 2011 and Casputi are three different comets	3
Een bezittelijk naamwoord	Whitaker discovered his comet in 2010	Whitaker's comet was discovered in 2010	1
Onbrekend woord (ellips)	The comet Parks discovered was discovered 1 year before Whitaker's	The comet Parks discovered was discovered 1 year before Whitaker's **comet**	1
Bijzin met foute structuur	The comet **Whitaker discovered** doesn't have an orbital period of 30 year	The comet **discovered by Whitaker** doesn't have an orbital period of 30 years	6
Naamwoordgroep heeft fout type	Gostroma was discovered **1 year** after the comet discovered by Tillman	Gostroma was discovered **1 cycle** after the comet discovered by Tillman	2

Tabel 9.9: Een overzicht van de aanpassingen aan logigram 19

Probleem	Origineel	Aangepast	Aantal
"the one"	Of the building haunted by Lady Grey and **the one** haunted by Victor, one was Markmanor and the other was visited in January	Of the building haunted by Lady Grey and **the building** haunted by Victor, one was Markmanor and the other was visited in January	1
Overtollig woord (said to be)	Wolfenden was **said to be** haunted by Brunhilde	Wolfenden was haunted by Brunhilde	2
Overtollig woord (Priscilla)	The house **Priscilla** visited in march wasn't located on Circle Drive	The house visited in march wasn't located on Circle Drive	3

Tabel 9.10: Een overzicht van de aanpassingen aan logigram 20

niet ondersteund zijn maar ook voorkwamen in de trainingset. In deze groep zijn er 5 aanpassingen.

Daarnaast is er een grote groep van aanpassingen die nodig zijn om te passen binnen het voorgestelde type-systeem. Dit gaat allemaal op overtredingen van het principe van één type per woord.

In één logigram is dan weer de assumptie overtreden dat elk domeinelement maar in één vorm voorkomt.

Ten slotte zijn er 10 aanpassingen nodig voor het aanvullen of weglaten van woorden die niet veel aan de essentie van de zin veranderen.

9.4 Vragen aan de gebruiker

Tabel 9.12 geeft een overzicht van het aantal vragen dat het systeem stelt aan de gebruiker. De tabel is ingedeeld per logigram en per soort vraag. Soort 1 is een vraag van de vorm

Probleem	Aantal
“the one”	15
Fout gebruik koppelwerkwoord	6
Bijzin met foute structuur	6
Passieve zin	1
Een bezittelijk voornaamwoord	1
Foute structuur naamwoordgroep	1
Kommagetallen	3
Superlatief	1
“Of the two ...”	1
Werkwoord heeft fout type	14
Naamwoordgroep heeft fout type	5
Dubbele vorm domeinelement	1
Overtollig woord	7
Ontbrekend woord (ellips)	3

Tabel 9.11: Een overzicht van alle aanpassingen

“Zijn A en B dezelfde relatie (synoniemen)?”. Soort 2 is van de vorm “Zijn A en B een omgekeerde relatie?”. Soort 3 ziet er uit als “Is A een mogelijk lijdend voorwerp van B?”. Soort 4 ten slotte heeft de vorm “Zijn A en B van hetzelfde type?”. De laatste vraag probeert het systeem te vermijden omdat dit het hele doel van de type-inferentie is.

De tabel geeft enkel de vragen met de positieve antwoorden omdat het resultaat dan niet afhankelijk is van de volgorde waarin de vragen gesteld worden.

Gemiddeld stelt het systeem per logigram twee vragen (met een positief antwoord) aan de gebruiker. Vragen van soort 1 dienen om synoniemen aan te geven die het systeem zelf niet kan afleiden. Vragen van soort 2 zijn analoog maar dan voor de omgekeerde relatie (“to give” en “to be given by”). Vragen van soort 3 en 4 zijn vaak nodig omwille van naamwoordgroepen met een onbekende relatie (zie sectie 6.3.2). Omdat de types van de verschillende eigennamen in zo’n naamwoordgroepen niet geünificeerd zijn, leidt elk van die naamwoordgroepen tot een vraag van soort 3 of 4. Daardoor komen deze soort vragen meestal in grotere aantallen voor.

Voor de meeste logigrammen heeft het systeem weinig vragen nodig. Er zijn vier logigrammen (logigram 8, 11, 13 en 17) die samen goed zijn voor de helft van de vragen.

9.5 Conclusie

Ten eerste kunnen we al stellen dat we elk logigram zodanig kunnen herschrijven zodat het mogelijk wordt om automatisch een volledige en correcte specificatie op te stellen. In de meeste gevallen zijn de aanpassingen eerder klein. Er zijn slechts twee zinnen die vrij hard aangepast moeten worden (één van passief naar actief en één voor het verwijderen van het bezittelijke voornaamwoord). Het is echter niet zo dat we de logigrammen gewoon kunnen ingeven zonder aanpassingen te doen. Daarvoor is de grammatica nog te ruim. Het lijkt echter wel mogelijk om een grammatica op te stellen waaraan een schrijver zich kan houden zonder dat de lezer een verschil kan merken.

Bovendien kunnen we ook altijd de types achterhalen. In de meeste gevallen enkel op basis van de drie soorten vragen die echt puur taalkundig zijn. Aan het principe van één

	Soort 1	**Soort 2**	**Soort 3**	**Soort 4**	**Totaal**
Logigram 1	1	0	0	0	**1**
Logigram 2	1	0	0	0	**1**
Logigram 3	0	0	0	0	**0**
Logigram 4	0	0	3	0	**3**
Logigram 5	0	0	1	0	**1**
Logigram 6	0	0	1	0	**1**
Logigram 7	1	0	1	0	**2**
Logigram 8	0	0	3	3	**6**
Logigram 9	0	0	3	0	**3**
Logigram 10	1	0	0	0	**1**
Logigram 11	0	0	0	4	**4**
Logigram 12	1	0	0	0	**1**
Logigram 13	1	0	4	0	**5**
Logigram 14	1	0	0	0	**1**
Logigram 15	0	1	0	0	**1**
Logigram 16	2	0	0	0	**2**
Logigram 17	1	0	3	0	**4**
Logigram 18	0	0	0	0	**0**
Logigram 19	1	0	0	0	**1**
Logigram 20	0	0	0	0	**0**
Totaal	**11**	**1**	**19**	**7**	**38**

Tabel 9.12: Een overzicht van het aantal vragen met een positief antwoord

type per woord is wel niet altijd voldaan. Daarvoor zijn er soms aanpassingen nodig.

Het resultaat is echter dat we op basis van het systeem beschreven in deze thesis de eerste twintig logigrammen uit [20] automatisch kunnen oplossen. Daarbij vertalen we eerst naar eerste-orde-logica. Op basis van de inferenties van de types kunnen we ook de nodige extra axioma's en het formeel vocabularium opstellen. Daardoor kunnen we met behulp van de tool IDP [10] de bijecties achterhalen en de domeinelementen opdelen in hun equivalentieklassen.

Hoofdstuk 10

Conclusie

Het is mogelijk om logigrammen automatisch op te lossen door ze te vertalen naar logica. De grootste moeilijkheid daarbij is het opstellen van een algemene grammatica die toepasbaar is op nieuwe logigrammen. Hoewel logigrammen reeds een beperkt aantal grammaticale regels volgen, zijn er toch nog aanpassingen nodig aan de zinnen om ze binnen de grammatica te krijgen.

Het framework van Blackburn en Bos [5, 6] is echter uitermate geschikt voor deze vertaling. Men kan nieuwe grammaticale en lexicale categorieën toevoegen die enkel van toepassing zijn binnen bepaalde domeinen, zoals de onbepaalde woorden dat zijn voor logigrammen. De betekenis van zinnen wordt via het compositionaliteitsprincipe teruggebracht tot de betekenis van woorden. Wanneer alle lexicale categorieën hun betekenis hebben, is het grootste deel van de vertaling naar logica gebeurd. De betekenis van de grammaticale regels bestaat meestal enkel uit één of meerdere lambda-applicaties.

De keuze van deze vertalingen bepaalt ook hoe bepaalde ambiguïteiten in taal worden opgelost. Zo zal in deze thesis een quantifier scope ambiguïteit opgelost worden door de kwantoren te rangschikken volgens hoe ze voorkomen in de zin in natuurlijke taal. Omdat hier een regelgebaseerd systeem is gebruikt, zijn deze keuzes ook deterministisch en kan de gebruiker ze ook leren. Dit maakt het gebruik in mission-critical systemen mogelijk.

Een beperking van het framework van Blackburn en Bos is dat ze grammaticaal correcte maar betekenisloze zinnen toch toestaan en proberen te vertalen naar logica. We willen echter niet dat in een specificatie zo'n zin kan voorkomen. Daarom hebben we een uitbreiding op het framework voorgesteld dat dit soort zinnen uitsluit. We hebben deze uitbreiding gebruikt om de verschillende domeinen van een logigram automatisch af te leiden. Voor een logigram met een beperkte woordenschat, lukt dit automatisch. Indien een logigram veel verschillende woorden gebruikt om hetzelfde te zeggen, kan het zijn dat de gebruiker moet helpen. Meestal kan dit puur op basis van taalkundige informatie. Indien er echter veel gebruik wordt gemaakt van naamwoordgroepen met een onbekende relatie (zie sectie 6.3.2), dan kan het zijn dat men de gebruiker moet vragen of twee domeinelementen tot hetzelfde domein behoren.

Verder onderzoek kan eruit bestaan om de besproken techniek toe te passen op andere domeinen. Bijvoorbeeld op andere soorten puzzels of echte specificaties. Daarnaast kan men ook een echte gecontroleerde natuurlijke taal voor logigrammen opstellen die ook

makkelijk geleerd kan worden en niet afgeleid is van bestaande logigrammen. Op die manier kan de taal simpel gehouden worden.

Ten slotte zijn ook veel vragen omtrent het type-checken van natuurlijke taal. Bijvoorbeeld of dit het aantal fouten kan verminderen of de productiviteit tijdens het schrijven kan verhogen. Daarnaast is er ook de vraag of er een nood is aan een ingewikkelder type-systeem en hoe dit systeem er dan zou uitzien. Aan het principe van één type per woord is vaak niet voldaan binnen logigrammen. In een systeem met type-checking kan men echter makkelijker meerdere types per woord toelaten. Het type-systeem kan dan het onderscheid maken tussen deze verschillende betekenissen. Zo kan men via type-checking controleren of in een zin "bestellen" gebruikt wordt om eten te bestellen of om drank te bestellen.

Er zijn al vele gecontroleerde talen opgesteld met heel uiteenlopende doelen. Binnen de (computationele) linguïstiek is er ook al veel werk verricht rond de betekenis van taal. Deze thesis toont aan dat dit onderzoek ook binnen het domein van kennisrepresentatie nuttig is. Meer zelfs, de computerwetenschappen kunnen dit onderzoek verder stimuleren door een aantal concepten uit de computerwetenschappen, zoals types, toe te voegen aan de bestaande frameworks voor het vertalen van natuurlijke taal. In deze thesis werd een aanzet gegeven hiervoor maar er is nog veel meer onderzoek nodig hieromtrent.

Bijlage A

Specificatie van logigram 1

In deze appendix geven we de specificatie die het systeem genereert voor logigram 1 uit Puzzle Baron's Logic Puzzles Volume 3 [20]. De zinnen van de logigram zien er uit als volgt:

1. The perens pig lives in Slovakia and was recognized as endangered in 2009
2. The eldar elk has a population size of 210
3. The byengo bat doesn't live in Ghana
4. The animal that lives in Slovakia was listed sometime before the animal from Russia
5. Of the species with a population size of 490 and the eldar elk, one lives in Poland and the other was recognized as endangered in 2009
6. Neither the nibner newt nor the byengo bat has a surviving population size of 525
7. The species that lives in Ghana was listed 2 years after the nibner newt
8. The animal that lives in Russia doesn't have a surviving population size of 315

Het lexicon voor deze logigram bevat

- 5 zelfstandige naamwoorden ("species", "animal", "year", "population size" en "surviving population size")
- 8 eigennamen ("perens pig", "byengo bat", "nibner newt", "eldar elk", "ghana", "poland", "russia" en "slovakia")
- 4 werkwoorden ("to have", "to live in", "to be recognized as endangered in" en "listed in").

Bij de type inferentie stelt het systeem de volgende vragen, enkel op de laatste vraag is het antwoord "ja".

1. Are 'recognized_as_endangered_in' and 'lives_in' the same relation? [yes/no]
2. Are 'recognized_as_endangered_in' and 'of' the same relation? [yes/no]
3. Are 'recognized_as_endangered_in' and 'has' the same relation? [yes/no]
4. Are 'recognized_as_endangered_in' and 'from' the same relation? [yes/no]
5. Are 'recognized_as_endangered_in' and 'listed_in' the same relation? [yes/no]

Bij het opstellen van het vocabularium stelt het ook nog de volgende twee vragen. De antwoorden zijn respectievelijk {2006, 2007, 2008, 2009, 2010} en {210, 280, 315, 490, 525}.

1. What are the possible values for year (e.g. the object of 'recognized_as_endangered_in')?
2. What are the possible values for type2 (e.g. the object of 'of')?

De oplossing die het systeem genereert is correct en ziet er uit als volgt

the_nibner_newt	the_other_type1	population_size_3	2006	315
the_eldar_elk	poland	population_size_1	2007	210
the_other_animal	ghana	population_size_5	2008	525
the_perens_pig	slovakia	population_size_4	2009	490
the_byengo_bat	russia	population_size_2	2010	280

Tabel A.1: De oplossing van logigram 1

Het systeem genereert die oplossing aan de hand van de IDP [10] specificatie hieronder

```
// Logigram 1

vocabulary V {
    type year = {2006; 2007; 2008; 2009; 2010} isa int
    type animal constructed from {
      the_other_animal,
      the_perens_pig,
      the_eldar_elk,
      the_byengo_bat,
      the_nibner_newt
    }
    type type1 constructed from {
      the_other_type1,
      slovakia,
      ghana,
      russia,
      poland
    }
    type type2 = {210; 280; 315; 490; 525} isa int
    type population_size constructed from {
      population_size_1,
      population_size_2,
      population_size_3,
      population_size_4,
      population_size_5
    }
    // differences between values of type year
    type type3 = {-1; 1; -2; 2; -3; 3; -4; 4} isa int

    recognized_as_endangered_in(animal, year)
    lives_in(animal, type1)
    of(population_size, type2)
    has(animal, population_size)
    from(animal, type1)
    listed_in(animal, year)
    with(animal, population_size)
}

structure S : V {
}
```

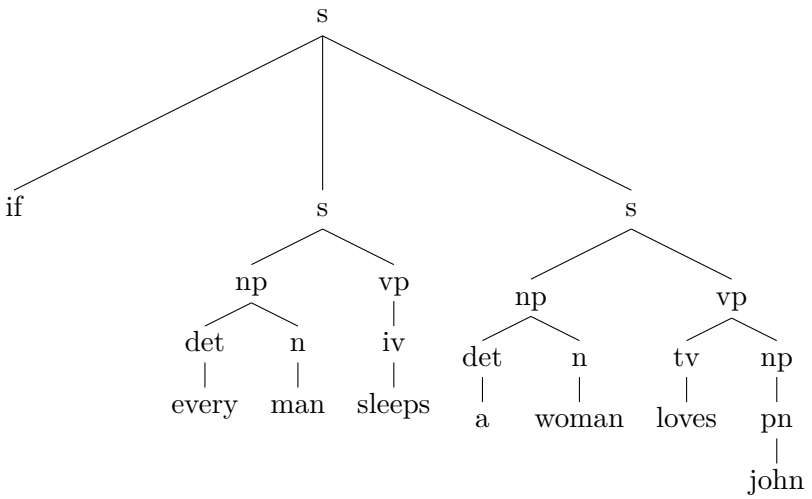
```
41
42 theory T : V {
43     // The perens pig lives in Slovakia and
44     // was recognized as endangered in 2009
45     lives_in(the_perens_pig,slovakia) &
46     recognized_as_endangered_in(the_perens_pig,2009).
47
48     // The eldar elk has a population size of 210
49     ?a [population_size]: of(a,210) & has(the_eldar_elk,a).
50
51     // The byengo bat doesn't live in Ghana
52     ~ lives_in(the_byengo_bat,ghana).
53
54     // The animal that lives in Slovakia was listed
55     // sometime before the animal from Russia
56     ?b [animal] c [type3] d [year] e [animal] f [year]:
57     lives_in(b,slovakia) & c>0 & from(e,russia) &
58     listed_in(e,d) & f = d-c & listed_in(b,f).
59
60     // Of the species with a population size of 490
61     // and the eldar elk, one lives in Poland and the other
62     // was recognized as endangered in 2009
63     ?g [animal] h [population_size]: of(h,490) &
64     with(g,h) & ~ (g = the_eldar_elk) &
65     (lives_in(g,poland) & recognized_as_endangered_in(the_eldar_elk,2009) |
66     lives_in(the_eldar_elk,poland) & recognized_as_endangered_in(g,2009)).
67
68     // Neither the nibner newt nor the byengo bat
69     // has a surviving population size of 525
70     ~ (?i [population_size]: of(i,525) & has(the_nibner_newt,i)) &
71     ~ (?j [population_size]: of(j,525) & has(the_byengo_bat,j)).
72
73     // The species that lives in Ghana was
74     // listed 2 years after the nibner newt
75     ?k [animal] l [year] m [year]: lives_in(k,ghana) &
76     listed_in(the_nibner_newt,l) & m = l+2 & listed_in(k,m).
77
78     // The animal that lives in Russia
79     // doesn't have a surviving population size of 315
80     ?n [animal]: lives_in(n,russia) &
81     ~ (?o [population_size]: of(o,315) & has(n,o)).
82
83
84     // Logigram bijection axioms:
85     ! x [animal]: ?=1 y [year]: recognized_as_endangered_in(x, y).
86     ! x [year]: ?=1 y [animal]: recognized_as_endangered_in(y, x).
87
88     ! x [animal]: ?=1 y [type1]: lives_in(x, y).
89     ! x [type1]: ?=1 y [animal]: lives_in(y, x).
90
```

```
91      ! x [population_size]: ?=1 y [type2]: of(x, y).
92      ! x [type2]: ?=1 y [population_size]: of(y, x).
93
94      ! x [animal]: ?=1 y [population_size]: has(x, y).
95      ! x [population_size]: ?=1 y [animal]: has(y, x).
96
97      ! x [animal]: ?=1 y [type1]: from(x, y).
98      ! x [type1]: ?=1 y [animal]: from(y, x).
99
100     ! x [animal]: ?=1 y [year]: listed_in(x, y).
101     ! x [year]: ?=1 y [animal]: listed_in(y, x).
102
103     ! x [animal]: ?=1 y [population_size]: with(x, y).
104     ! x [population_size]: ?=1 y [animal]: with(y, x).
105
106     // Logigram synonym axioms:
107     ! x [animal] y [year]: recognized_as_endangered_in(x, y) <=> listed_in(x, y).
108     ! x [animal] y [type1]: lives_in(x, y) <=> from(x, y).
109     ! x [animal] y [population_size]: has(x, y) <=> with(x, y).
110
111     // Logigram transitivity axioms:
112
113     // Logigram symmetry axioms:
114
115     // Logigram symmetry breaking axioms:
116     of(population_size_1, 210).
117     of(population_size_2, 280).
118     of(population_size_3, 315).
119     of(population_size_4, 490).
120     of(population_size_5, 525).
121 }
122
123 procedure main() {
124     // Search for 10 possible solutions to the logigram.
125     // It should and does only find one
126     stdoptions.nbmodels = 10;
127     printmodels(modelexpand(T,S))
128 }
```

Bijlage B

Een illustratie van het semantische framework

In deze appendix illustreren we het framework van Blackburn en Bos (zie hoofdstuk 4) aan de hand van de zin "If every man sleeps, a woman loves John". De parse tree die bij deze zin hoort is



Frege's compositionaliteitsprincipe leert ons dat we elke knoop apart mogen behandelen om daarna de resultaten te combineren. In de rest van deze appendix, doorlopen we alle knopen in de boom. We passen de formules die we hierboven hebben afgeleid toe en vereenvoudigen vervolgens met behulp van beta-reductie uit de lambda-calculus.

B.1 Every man sleeps

$$[\![np_{every\ man}]\!] = [\![det]\!]([\![n]\!])$$

$$= \left(\lambda R \cdot \lambda S \cdot \boxed{\left\{ \boxed{x} \oplus R(x) \right\} \Rightarrow S(x)} \right) \left(\lambda y \cdot \boxed{\boxed{man(y)}} \right)$$

$$= \lambda S \cdot \boxed{\left\{ \boxed{\begin{array}{|c|}\hline x \\ \hline \\ \hline \end{array}} \oplus \left(\lambda y \cdot \boxed{\begin{array}{|c|}\hline \\ \hline man(y) \\ \hline \end{array}} \right) (x) \right\} \Rightarrow S(x)}$$

$$= \lambda S \cdot \boxed{\left\{ \boxed{\begin{array}{|c|}\hline x \\ \hline \\ \hline \end{array}} \oplus \boxed{\begin{array}{|c|}\hline \\ \hline man(x) \\ \hline \end{array}} \right\} \Rightarrow S(x)}$$

$$= \lambda S \cdot \boxed{\boxed{\begin{array}{|c|}\hline x \\ \hline man(x) \\ \hline \end{array}} \Rightarrow S(x)}$$

$$[\![s_{every\ man\ sleeps}]\!] = [\![vp]\!]([\![np]\!])$$

$$= \left(\lambda O \cdot O \left(\lambda y \cdot \boxed{\begin{array}{|c|}\hline \\ \hline sleeps(y) \\ \hline \end{array}} \right) \right) \left(\lambda S \cdot \boxed{\boxed{\begin{array}{|c|}\hline x \\ \hline man(x) \\ \hline \end{array}} \Rightarrow S(x)} \right)$$

$$= \left(\lambda S \cdot \boxed{\boxed{\begin{array}{|c|}\hline x \\ \hline man(x) \\ \hline \end{array}} \Rightarrow S(x)} \right) \left(\lambda y \cdot \boxed{\begin{array}{|c|}\hline \\ \hline sleeps(y) \\ \hline \end{array}} \right)$$

$$= \boxed{\boxed{\begin{array}{|c|}\hline x \\ \hline man(x) \\ \hline \end{array}} \Rightarrow \left(\lambda y \cdot \boxed{\begin{array}{|c|}\hline \\ \hline sleeps(y) \\ \hline \end{array}} \right) (x)}$$

$$= \boxed{\boxed{\begin{array}{|c|}\hline x \\ \hline man(x) \\ \hline \end{array}} \Rightarrow \boxed{\begin{array}{|c|}\hline \\ \hline sleeps(x) \\ \hline \end{array}}}$$

B.2 A woman loves John

$$[\![np_{a\ woman}]\!] = [\![det]\!]([\![n]\!])$$

$$
\begin{aligned}
&= \left(\lambda R \cdot \lambda S \cdot \left\{ \boxed{\begin{array}{|c|}\hline x \\ \hline \\ \hline \end{array}} \oplus R(x) \oplus S(x) \right\}\right) \left(\lambda y \cdot \boxed{\begin{array}{|c|}\hline \\ \hline woman(y) \\ \hline \end{array}}\right) \\
&= \lambda S \cdot \left\{ \boxed{\begin{array}{|c|}\hline x \\ \hline \\ \hline \end{array}} \oplus \left(\lambda y \cdot \boxed{\begin{array}{|c|}\hline \\ \hline woman(y) \\ \hline \end{array}}\right)(x) \oplus S(x) \right\} \\
&= \lambda S \cdot \left\{ \boxed{\begin{array}{|c|}\hline x \\ \hline \\ \hline \end{array}} \oplus \boxed{\begin{array}{|c|}\hline \\ \hline woman(x) \\ \hline \end{array}} \oplus S(x) \right\} \\
&= \lambda S \cdot \left\{ \boxed{\begin{array}{|c|}\hline x \\ \hline woman(x) \\ \hline \end{array}} \oplus S(x) \right\}
\end{aligned}
$$

$$
\begin{aligned}
[\![vp_{loves\ john}]\!] &= [\![tv]\!]([\![np]\!]) \\
&= \left(\lambda L \cdot \lambda O \cdot O\left(\lambda x_o \cdot L\left(\lambda x_l \cdot \boxed{\begin{array}{|c|}\hline \\ \hline loves(x_o, x_l) \\ \hline \end{array}}\right)\right)\right)(\lambda P \cdot P(john)) \\
&= \lambda O \cdot O\left(\lambda x_o \cdot (\lambda P \cdot P(john))\left(\lambda x_l \cdot \boxed{\begin{array}{|c|}\hline \\ \hline loves(x_o, x_l) \\ \hline \end{array}}\right)\right) \\
&= \lambda O \cdot O\left(\lambda x_o \cdot \left(\lambda x_l \cdot \boxed{\begin{array}{|c|}\hline \\ \hline loves(x_o, x_l) \\ \hline \end{array}}\right)(john)\right) \\
&= \lambda O \cdot O\left(\lambda x_o \cdot \boxed{\begin{array}{|c|}\hline \\ \hline loves(x_o, john) \\ \hline \end{array}}\right)
\end{aligned}
$$

Niet alleen heeft deze lambda-expressie een signatuur die gelijk is aan die van een onovergankelijk werkwoord, de structuur lijkt er ook sterk op.

$$
\begin{aligned}
[\![s_{a\ woman\ loves\ john}]\!] &= [\![vp]\!]([\![np]\!]) \\
&= \left(\lambda O \cdot O\left(\lambda x_o \cdot \boxed{\begin{array}{|c|}\hline \\ \hline loves(x_o, john) \\ \hline \end{array}}\right)\right)\left(\lambda S \cdot \left\{ \boxed{\begin{array}{|c|}\hline x \\ \hline woman(x) \\ \hline \end{array}} \oplus S(x) \right\}\right) \\
&= \left(\lambda S \cdot \left\{ \boxed{\begin{array}{|c|}\hline x \\ \hline woman(x) \\ \hline \end{array}} \oplus S(x) \right\}\right)\left(\lambda x_o \cdot \boxed{\begin{array}{|c|}\hline \\ \hline loves(x_o, john) \\ \hline \end{array}}\right) \\
&= \left\{ \boxed{\begin{array}{|c|}\hline x \\ \hline woman(x) \\ \hline \end{array}} \oplus \left(\lambda x_o \cdot \boxed{\begin{array}{|c|}\hline \\ \hline loves(x_o, john) \\ \hline \end{array}}\right)(x) \right\} \\
&= \left\{ \boxed{\begin{array}{|c|}\hline x \\ \hline woman(x) \\ \hline \end{array}} \oplus \boxed{\begin{array}{|c|}\hline \\ \hline loves(x, john) \\ \hline \end{array}} \right\} \\
&= \boxed{\begin{array}{|l|}\hline x \\ \hline woman(x) \\ loves(x, john) \\ \hline \end{array}}
\end{aligned}
$$

B.3 If every man sleeps, a woman loves John

$$
[\![s]\!] = \boxed{\begin{array}{|c|}\hline \\ \hline [\![s1]\!] \Rightarrow [\![s2]\!] \\ \hline \end{array}}
$$

$$= \left(\forall x \cdot man(x) \Rightarrow sleeps(x)\right) \Rightarrow \left(\exists y \cdot woman(y) \wedge loves(y, john)\right)$$

Dit is de vertaling zoals we die zouden verwachten.

Bibliografie

[1] V. Ambriola and V. Gervasi. Processing natural language requirements. *Proceedings 12th IEEE International Conference Automated Software Engineering*, pages 36–45, 1997.

[2] C. Baral and J. Dzifcak. Solving Puzzles Described in English by Automated Translation to Answer Set Programming and Learning How to Do that Translation. *Knowledge Representation and Reasoning (KR)*, pages 573–577, 2012.

[3] C. Baral, J. Dzifcak, and T. C. Son. Using Answer Set Programming and Lambda Calculus to Characterize Natural Language Sentences with Normatives and Exceptions. *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI-08)*, pages 818–823, 2008.

[4] C. Baral, M. A. Gonzalez, and A. Gottesman. The inverse lambda calculus algorithm for typed first order logic lambda calculus and its application to translating english to FOL. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7265:40–56, 2012.

[5] P. Blackburn and J. Bos. Representation and inference for natural language, 2005.

[6] P. Blackburn and J. Bos. Working with discourse representation theory. *An Advanced Course in Computational Semantics*, 2006.

[7] P. Blackburn and K. Striegnitz. Natural language processing techniques in prolog. http://cs.union.edu/~striegnk/courses/nlp-with-prolog/html/index.html, 2002.

[8] J. Bos. A Survey of Computational Semantics: Representation, Inference and Knowledge in Wide-Coverage Text Understanding. *Linguistics and Language Compass*, 5(6):336–366, 2011.

[9] S. Costantini and A. Paolucci. Towards translating natural language sentences into asp. In *CILC*, 2010.

[10] B. de Cat, B. Bogaerts, M. Bruynooghe, and M. Denecker. Predicate logic as a modelling language: The IDP system. *CoRR*, abs/1401.6312, 2014.

[11] N. E. Fuchs, K. Kaljurand, and T. Kuhn. Attempto controlled english for knowledge representation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5224 LNCS:104–124, 2008.

[12] N. E. Fuchs and T. Kuhn. ACE 6.7 construction rules. http://attempto.ifi.uzh.ch/site/docs/ace_constructionrules.html.

[13] T. Kuhn. A Survey and Classification of Controlled Natural Languages. *Computational Linguistics*, 40(1):121–170, mar 2014.

[14] F. Lévy and A. Nazarenko. Formalization of natural language regulations through sbvr structured english (tutorial). *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8035 LNCS:19–33, 2013.

[15] M. Luisa, F. Mariangela, and N. I. Pierluigi. Market Research for Requirements Analysis Using Linguistic Tools. *Requirements Engineering*, 9(1):40–56, 2004.

[16] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[17] P. B. F. Njonko, S. Cardey, P. Greenfield, and W. El Abed. RuleCNL: A controlled natural language for business rule specifications. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8625 LNAI:66–77, 2014.

[18] F. Pereira and D. Warren. Definite Clause Grammar for Language Analysis - a survey of the formalism and a comparison with ATN. *Artificial Intelligence*, 13(2):231–278, 1980.

[19] R. G. Ross. RuleSpeak Sentence Forms. *RuleSpeak*, page 10, 2009.

[20] S. Ryder. *Puzzle Baron's logic puzzles*. Alpha Books, Indianapolis, Indiana, 2016.

[21] R. Schwitter. English as a Formal Specification Language. *Interpreting*, pages 228–232, 2002.

[22] R. Schwitter. Representing knowledge in controlled natural language: a case study. *Knowledge-Based Intelligent Information and Engineering Systems*, pages 711–717, 2004.

[23] R. Schwitter, A. Ljungberg, and D. Hood. ECOLE–A Look-ahead Editor for a Controlled Language. *Eamt-Claw03*, pages 141–150, 2003.

[24] R. Schwitter and M. Tilbrook. Controlled Natural Language meets the Semantic Web. *Proceedings of the Australasian Language Technology Workshop*, pages 55–62, 2004.

[25] R. Schwitter, M. Tilbrook, et al. Let's talk in description logic via controlled natural language. 2006.

[26] Shieber and M. Stuart. *An Introduction to Unification-Based Approaches to Grammar*. Microtome Publishing, 2003.

[27] Wikipedia. Zebra Puzzle — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Zebra%20Puzzle&oldid=779870856, 2017. [Online; accessed 24-May-2017].

Fiche masterproef

Student: Jens Claes

Titel: Automatisch vertalen van logigrammen naar een getypeerde logica

Engelse titel: Automatic Translation of Logic Grid Puzzles into a Typed Logic

UDC: 681.3

Korte inhoud:
Deze thesis evalueert en verbetert het semantische framework van Blackburn en Bos [5, 6]. Specifiek passen we dit framework toe op het vertalen van logigrammen naar logica. Hiervoor stellen we een set van lexicale categorieën en een grammatica op, specifiek voor logigrammen, op basis van tien logigrammen. We evalueren het framework op tien nieuwe logigrammen. Hierbij onderzoeken we of de nieuwe logigrammen, mits aanpassingen, uitdrukbaar zijn in de opgestelde grammatica en wat voor aanpassingen dan nodig zijn. Verder breiden we het framework uit met types. Dankzij types kunnen grammaticaal correcte zinnen zonder betekenis, zoals “Het gras drinkt het zingende huis”, toch uitgesloten worden. Binnen deze thesis worden de types gebruikt om de verschillende domeinen van een logigram af te leiden. Daarnaast staan types ons toe om te vertalen naar een getypeerde logica.

Thesis voorgedragen tot het behalen van de graad van Master of Science in de ingenieurswetenschappen: computerwetenschappen, hoofdspecialisatie Artificiële intelligentie

Promotor: Prof. dr. M. Denecker

Assessoren: Prof. dr. M.-F. Moens
Dr. B. Bogaerts

Begeleiders: Dr. B. Bogaerts
Ir. L. Janssens

Automatic Translation of Logic Grid Puzzles into a Typed Logic

Jens Claes

***Abstract*—This paper extends and evaluates the semantical framework of Blackburn and Bos for knowledge representation. Specifically, the framework is used to translate logic grid puzzles into logic.**

We introduce types into the framework to translate to a typed logic and to allow the system to infer the domains of a logic grid puzzle from its sentences in natural language. This extension also enables type checking of sentences in natural language. Therefore, it can reject sentences that are grammatical but without meaning.

I. Introduction

KNOWLEDGE base systems have been around for a while. They take a knowledge base as input. This knowledge base describes the world, i.e. it is a specification of how something should work. Based on this knowledge base, different kinds of inference can be applied. De Cat et al. [1] give the example of an university course-management system. The knowledge base contains rules like "In every auditorium at a any time, there should be at most one course.". The different kinds of inferences that can be applied, include (the examples are from De Cat et al. [1]): *propagation* (e.g. automatically selecting required prerequisites when constructing an individual study program), *model expansion* (e.g. getting a full individual study program from a partial one) and *querying* (e.g. getting the schedule for a specific student).

These knowledge base systems have triggered a lot of research into formal languages and their expressivity. These languages are often hard to read, write and learn.

One method to ease the writing of formal theories is to develop languages that are somewhere in between natural language and formal languages. Such languages are called Controlled Natural Languages (CNL). A CNL is a subset of a natural language that, for example, allows to write specifications in a more consistent language. Kuhn [2] made an overview of 100 CNL's. Some of these languages have formal semantics and can be translated automatically into a formal logic. They can be used as a knowledge representation language within a knowledge base system. They are often easy to read. Unfortunately, the translations of such CNL's into logic are not well documented which makes it hard to expand the language.

This paper therefore constructs a new CNL based on the language used in a small domain, namely logic grid puzzles. It is then tested on unseen puzzles. The first goal of this paper is to show that the framework introduced by Blackburn and Bos [3], [4] can be used to automatically translate such a CNL into a formal logic. The second goal is to show that adding types to this framework is useful. It makes it possible to translate to a typed logic and allows more inference. E.g. The system can automatically infer the domains of a logic grid puzzle.

Therefore, we take the next step for bridging the gap between natural language and logic. We extend the well documented framework of Blackburn and Bos with types. With this extension, we constructed the first typed CNL with a formal semantics in a typed logic.

The result of this research is a fully automated tool that can understand a logic grid puzzle and reason about it (e.g. the system can solve the puzzle automatically).

II. Logic Grid Puzzles

Logic grid puzzles are logical puzzles. The most famous such puzzle is probably the Zebra Puzzle [5], sometimes also known as Einstein's Puzzle. These puzzles consist of a number of sentences or clues in natural language. In the clues, a number of domains appear (e.g. nationality, color, animal, ...), each with a number of domain elements (e.g. Norwegian, Canadian, blue, red, cow, horse, ...). Between each domain there is a bijection. The goal of such a puzzle is to find the value of these bijections, i.e. to find which domain elements belong together. E.g. The Norwegian lives in a blue house and keeps the horse. Every puzzle has an unique solution.

Logic grid puzzles can be considered as small specifications. Moreover, they can be expressed in fairly simple logical statements. Finally, it is easy to find numerous examples of these types of puzzles. It is these three properties that make logic grid puzzles the ideal domain to test our assumptions. Namely that the framework of Blackburn and Bos allows to construct a CNL with formal semantics that can be used for knowledge representation (within a small domain).

III. Motivation

Since knowledge representation languages can be hard to read, write and learn, an expert in formal languages is required to write and maintain the specification. A formal language that is more accessible for non-experts, like a CNL with formal semantics, can solve this issue without compromising on the strength of knowledge base systems. It is still possible to apply different kind of inferences on a specification that is more accessible for non-experts.

For example, when the clues of a logic grid puzzle are expressed in a CNL (with formal semantics), it is possible to automatically construct a complete specification such that a knowledge base system can reason about the puzzle. Such a system can

- Solve the puzzle automatically.
- Given a (partial) solution by the user, indicate which clues have already been incorporated in the solution, which clues still hold some new information and which clues have been violated.
- Given a partial solution by the user, automatically derive a subset of clues that can be used to expand the solution. This can be part of a "hint"-system for the user.
- Given a set of clues, indicate which solutions are still possible. This can help the author while writing new puzzles. Based on the solutions that are still possible, the author can construct a new clue to eliminate some of the possibilities until only one solution remains.

IV. An overview of the system

We created such a system. It can reason about a logic grid puzzle starting from the clues in (controlled) natural language and a puzzle-specific lexicon. The system uses the following steps

1) It translates the clues into logic using the extended framework of Blackburn and Bos. While doing so, it gathers type information about the words in the lexicon.
2) The system uses this type information to automatically deduce the domains of the puzzle. If necessary, it asks the user questions.
3) The system then constructs a formal vocabulary based on these domains.
4) Thereafter, it formulates the necessary axioms to model the implicit assumptions of a logic grid puzzle
5) Finally, the system passes the complete specification (consisting of the formal vocabulary and the theory with the axioms and the translation of the clues) to the IDP system [1] to get a solution to the puzzle (or to apply another kind of inference)

For this system, we extended the framework of Blackburn and Bos, we constructed a grammar and a set of lexical categories that can be used in this framework for the translation of logic grid puzzles into logic. We devised an inference system to deduce the different domains from the type information. We contrived how to express these domains in a formal vocabulary and finally, we listed the axioms necessary to complete the specification.

V. A semantical framework

Blackburn and Bos constructed a framework [3], [4] to capture the meaning of natural language by translating it into a logic. It consists of four parts: the lexicon, the grammar, the semantics of the lexicon and the semantics of the grammar. The framework is based on the λ-calculus and Frege's compositionality principle. Every word has a λ-expression as its meaning. The meaning of a group of words is a combination of the meaning of the words that are part of the group. In this framework, λ-application is used to combine the meaning of words.

The grammar defines which sentences are grammatically correct (and thus allowed) and which are not. E.g. a correct sentence consists of a noun phrase and a verb phrase that agree in number. This process results in a parse tree of every grammatically correct sentence, in which the words are the leaves. The framework then combines the meaning of the leaves (the words) upwards in the tree to the meaning of the root node (the sentence).

```
1  s([]) -->
2    np([num:Num]),
3    vp([num:Num]).
```

```
4  np([num:sg]) -->
5    pn([]).
6  np([num:Num]) -->
7    det([]),
8    n([num:Num]).
9  vp([num:Num, sem:Sem]) -->
10   iv([num:Num, sem:Sem]).
11 vp([num:Num, sem:Sem]) -->
12   tv([num:Num, sem:]),
13   np([num:_]).
```

Grammar 1: A simple grammar

The lexicon consists of the enumeration of the different words that can be used along with some linguistic features. E.g. "loves" is a transitive verb in the present tense, "love" is a transitive verb and an infinitive. A lot of words also have a feature *Symbol* which is used in the translation as the name of a constant or predicate.

The goal of the framework is to lexicalize as much of the meaning as possible. The semantics of a grammar rule should be limited to λ-applications, if possible. Table I gives the semantics for grammar 1.

Grammatical rule	Semantics
S (line 1-3)	$[\![s]\!] = [\![vp]\!]([\![np]\!])$
NP1 (line 4-5)	$[\![np]\!] = [\![pn]\!]$
NP2 (line 6-8)	$[\![np]\!] = [\![det]\!]([\![n]\!])$
VP1 (line 9-10)	$[\![vp]\!] = [\![iv]\!]$
VP2 (line 11-13)	$[\![vp]\!] = [\![tv]\!]([\![np]\!])$

TABLE I: The semantics of grammar 1

The meaning of the words themselves is the only thing still missing. Blackburn and Bos assume that the meaning of a word is only dependent on its linguistic features, most importantly its lexical category. Constructing these λ-expressions can be hard. Therefore, it can help to analyze the signature of these expressions. Blackburn and Bos suggest a λ-calculus with two types for this: e represents entities and t truth-values. The signature of a noun is $\tau(n) = e \to t$, given an entity, the noun says whether or not the entity can be described with the noun. E.g. $[\![n_{man}]\!] = \lambda x \cdot man(x)$

A noun phrase represents one or more entities. E.g. "a man", "every woman". A sentence is true if enough entities of the noun phrase satisfy a certain property of the verb phrase. The signature of a noun phrase is thus $\tau(np) = (e \to t) \to t$. Given a property P with signature $e \to t$, the semantics of the noun phrase tell us if the noun phrase satisfies the property P of the verb phrase. The simplest noun phrase is a proper noun like John. Its semantics is $[\![pn_{John}]\!] = \lambda P \cdot P(John)$. I.e. when we say something about "John", the property should hold for $John$. The meaning of the noun phrase "every man" is $[\![np_{every\ man}]\!] = \lambda P \cdot \forall x \cdot man(x) \Rightarrow P(x)$. The property P should hold for all men for the sentence to be true. From this, we conclude the meaning of "every" as $[\![det_{universal}]\!] = \lambda R \cdot \lambda P \cdot \forall x \cdot R(x) \Rightarrow P(x)$. We call R the restriction by the noun. We can generalize this to other determiners like "two": $[\![det_{binary}]\!] = \lambda R \cdot \lambda P \cdot \exists_{>2} x \cdot R(x) \wedge P(x)$.

A verb phrase should say whether the sentence is true or not, given a noun phrase as its subject. An intransitive verb like "sleeps" is the simplest verb phrase. Its meaning is $[\![iv_{sleeps}]\!] = \lambda S \cdot S(\lambda x \cdot sleeps(x))$. The sentence is true if the subject S holds the property $\lambda x \cdot sleeps(x)$. In other words, the sentence is true, if the semantics of the subject S (a function from a property with signature $e \to t$ to a truth value) evaluates to $true$ for the property $\lambda x \cdot sleeps(x)$.

The meaning of the sentence "John sleeps" is then

$$[\![s]\!] = [\![vp]\!]([\![np]\!]) =$$

$$(\lambda S \cdot S(\lambda x \cdot sleeps(x)))(\lambda P \cdot P(John))$$

$$= sleeps(John)$$

Finally, the meaning of a transitive verb like "loves" is

$$[\![tv_{loves}]\!] = \lambda O \cdot \lambda S \cdot S(\lambda x_S \cdot O(\lambda x_O \cdot loves(x_S, x_O)))$$

An entity x_S from the subject S satisfies the property if there are enough corresponding x_O's from the object O such that $loves(x_S, x_O)$.

For example for the sentence "Every man loves a woman", a man x_S satisfies the verb phrase ("loves a woman") if for this man x_S, there is a woman x_O such that $loves(x_S, x_O)$. The subject (as a whole) satisfies the verb phrase if *every* man x_S satisfies the verb phrase, i.e. if there is a woman x_O for every man x_S such that the man loves the woman. This woman can be different for every man.

This translation therefore decides how quantifier scope ambiguities are resolved, namely the translation follows the order of the natural language. This is how all quantifier scope ambiguities are resolved within this paper. Blackburn and Bos explain in their books how all readings can be obtained.

A. A lexicon for logic grid puzzles

To use the framework for translating logic grid puzzles into logic, we constructed a set of 12 lexical categories. These categories are determiner, number, noun, proper noun, preposition, relative pronoun, transitive verb, auxiliary verb, copular verb, comparative, *some*-words (somewhat, sometime, ...) and conjunction. Because of space constraints, we refer the reader to the full master thesis for the λ-expression of these categories.

The determiners in logic grid puzzles we studied are simple in that only existential quantifiers are necessary. Universal quantification is not needed as the puzzles always fully classify their noun phrases. This is a consequence of the fact that all relations described in these puzzles are bijections between the different domains. There is always exactly one person who, for example, drinks tea. For the same reason there is also no negative quantifier. Sentences like "No person drinks tea" do not occur in the puzzles we studied.

Proper nouns play an important role in our translation of logic grid puzzles into logic. We assume that every domain element (of a non-numeric) domain is represented by a proper noun (and only one). Even when the words are adjectives, they must be declared as proper nouns to the system. Proper nouns can also be numeric, e.g. "March". In that case the user has to encode it into a number.

There are three words that introduce a predicate in their translation in logic. These predicates express a relation between two entities. A transitive verb links its subject and its object. A preposition (in a prepositional phrase) links the noun phrase of the prepositional phrase to the noun phrase to which it is attached. Finally, an adjective phrase as object of a copular verb, links the subject with the noun phrase that is part of the adjective phrase. This is the only place an adjective can occur in our grammar.

Finally there are two lexical categories which can be considered special to logic grid puzzles: comparatives and *some*-words. The latter are, in the puzzles we studied, only used as an unspecified integer and always appear as a quantity in a comparison (e.g. "somewhat more than John").

B. A grammar for logic grid puzzles

A grammar was constructed based on the first ten logic grid puzzles from Puzzle Baron's Logic Puzzles Volume 3 [6].

In total there are 49 grammar rules: 3 grammar rules for sentences, 20 related to nouns and noun phrases, 8 for verb phrases and 18 rules that connect the grammar with the lexicon.

Again, because of space constraints, we cannot show the full grammar. We limit ourselves to a short discussion.

Most grammar rules are quite general. They can be used outside logic grid puzzles as well. Others, like the grammar rule for a sentence with template "Of ... and ..., one ... and the other ...", are specific for these types of puzzles. The introduction to [6] explicitly mentions this template and explains how this template should be interpreted. This interpretation is incorporated in the semantics of the grammar rule covering this template.

Some other more specific rules include an *all-different* constraint ("A, B, and C are three different persons") and a numerical comparison ("John scored 3 points higher than Mary") which introduces an addition or subtraction.

The semantics of most rules consists only of λ-applications. Some rules are more complex. Those exceptions are either because the scope of variables and negations would otherwise be incorrect or because the grammar rule is specific to logic grid puzzles and can not be easily explained linguistically. This linguistic shortcut is then visible in the semantics.

There are two type of noun phrases which relate two entities without a reference to this relation. Namely "the 2008 graduate" and "John's dog". In the current framework, it is impossible to derive which relation holds between "the graduate" and "2008". In Section VII-A, we discuss a solution to this problem.

VI. Types

In natural language, it is possible to construct sentences that are grammatically correct but without meaning. E.g. "The grass is drinking the house". The grass is not something that can drink and a house is not something that can be drunk. We say the sentence is badly typed. Based on grammar alone, we can never exclude these sentences. Therefore, we add types to the framework.

Concretely, we add a feature *type* to most words and phrases to indicate their type. Similarly to the feature *number* that some words and phrases have. The feature *number* indicates whether a word or phrase is singular or plural. In a sentence, the *number* of the noun phrase and the verb phrase must unify, i.e. the subject and the verb must agree

in number. Similarly to this, the *type* of the noun phrase and the verb phrase must unify.

In this paper we use a very basic type system. There are a number of basic types and a *pair* type constructor that takes two basic types. E.g. a noun has a basic type, a transitive verb a pair of types: one for its subject and one for its object. Also phrases get types. A noun phrase gets a basic type. The same goes for a verb phrase. In the grammar we then express that the type of the noun phrase and the verb phrase should unify when they make a sentence. This excludes badly typed sentences from being accepted.

We also use these types in the semantics to indicate the types of the logical variables and of predicates. This allows the system to translate to a typed logic.

As indicated earlier, most words (like noun, verb, ...) also get a type in the lexicon. In this paper we explored a form of type inference. We assume that every word has exactly one type but that is not explicitly given to the system. We then search the types of all words. Based on these types, we can group the domain elements (represented by a proper noun) per domain (represented by a type), i.e. we can infer that "France" and "Italy" are from the same domain (without necessarily knowing that they are countries).

The search for the types of all words gets as input the number of domains in the puzzle (related to the number of base types) and some type constraints (from the translation of the clues into logic, based on the principle of one type per word). If many synonyms are used, the system enters an interactive mode and asks the user some linguistic questions that can be rewritten as "Is ... a meaningful sentence?" until the types are unified enough.

This process fails if (and only if) a certain domain always occurs as part of noun phrases with an unknown relation (e.g. "the 2008 graduate"). In such a scenario, the system cannot fallback on a linguistic question and therefore asks the user if two proper nouns are of the same type. This is the exact question the system was supposed to solve.

The described system can thus correctly derive which domain elements belong together based on type constraints and the answers to some linguistic questions. Only when they do not provide enough information, is the user asked to help identify the elements that belong together.

VII. A COMPLETE SPECIFICATION

In step 3 and 4 of the system described in IV, the system completes the specification with a formal vocabulary and the formulation of the implied axioms. We now describe these steps.

A. The formal vocabulary

We can construct the formal vocabulary based on linguistic information and type information.

A type that corresponds to a non-numerical domain of the puzzle is translated to a constructed type. A constructed type consist of a set of constants with two extra axioms implied: Domain Closure Axiom (DCA) and Unique Names Axiom (UNA). DCA states that the set of constants are the only possible elements of the domain. UNA states that all constants are different from each other.

The different proper nouns that belong to this type are the constants of the constructed type. However, it is possible that one constant is missing (i.e. it didn't occur in any of the clues). In that case, the system adds an extra constant. It is not possible that two elements are missing, because every puzzle has an unique solution and these two elements would be interchangeable.

A type that corresponds to a numerical domain is translated to a subset of the natural numbers. In that case, the system asks the user the exact subset as it often cannot be automatically inferred from the clues.

It is also possible to have an intermediate type that links two domains, e.g. *tour* in "John follows the tour with 54 people". These types are translated to constructed types with as many constants as there are domain elements in a domain. Symmetry-breaking axioms are added to the theory to link every constant with one domain element of another domain.

Every transitive verb and preposition introduces a predicate. The type signature of the predicate corresponds with the type pair of the word. Based on these signatures, we can infer the relation behind noun phrases like "the 2008 graduate". The predicate that links the two entities is the predicate that links the corresponding types.

B. The implied axioms

There are some axioms implied for a logic grid puzzle. They are not expressed in the clues because they hold for all logic grid puzzles.

The first type of axioms is the symmetry-breaking axioms mentioned in VII-A for intermediate types (E.g. $with(T1, 54) \wedge with(T2, 64) \wedge with(T3, 74)$). Another type is the bijection axioms to state that every predicate is a bijection (E.g. $\forall x \cdot \exists y \cdot with(x, y) \wedge \forall y \cdot \exists x \cdot with(x, y)$).

Finally there are three types of axioms to express that there is an equivalence relation between domain elements. Two elements are equivalent if they are linked through a predicate or if they are equal. In second order logic $x \sim y \Leftrightarrow \exists P \cdot P(x, y) \vee x = y$. The reflexivity is satisfied by definition.

- **Synonymy** There is exactly one bijection between every two domains. Therefore, two predicates with the same types are always synonyms. E.g. $\forall x \forall y \cdot livesIn(x, y) \Leftrightarrow from(x, y)$.
- **Symmetry** Predicates with a reversed signature are each other inverse. E.g. $\forall x \forall y \cdot gave(x, y) \Leftrightarrow givenBy(y, x)$.
- **Transitivity** Finally there are axioms linking the different predicates. They express the transitivity of the equivalence relation. E.g. $\forall x \forall y \cdot spokeFor(x, y) \Leftrightarrow \exists z \cdot gave(x, z) \wedge lastedFor(z, y)$.

VIII. Evaluation

A grammar was constructed based on the ten first logic grid puzzles from Puzzle Baron's Logic Puzzles Volume 3 [6] and evaluated on the next ten puzzles. Some rare and difficult constructs in the training set were modified to make it easier to parse. To fit the unseen puzzles to the constructed grammar, some clues of the test set were modified as well. The question arises how many of these modifications are necessary to represent the unseen puzzles in the constructed grammar. Another question is whether or not it is possible to deduce the types of a logic grid puzzle or not. Finally, we want to know if all puzzles are represented correctly such that the solution can automatically be derived with the IDP system [1].

It turns out that once modified to the grammar, the types of nine unseen puzzles can be derived automatically and the clues can be translated correctly into logic. For one puzzle the system asks the user if certain domain elements belong to the same domain. Thereafter, it does translate the puzzle correctly into logic.

Table II gives an overview of the different modifications. In total 65 modifications were necessary, 30 of which were purely grammatical. They used grammatical structures that didn't appear in the training set. The next 5 modifications were modifications similar to those from the training set. There are 19 modifications due to badly typed words. E.g. a verb "to order" that was used to order both food and drinks. The assumption of one type per word wasn't satisfied in that case. There was one modification to a proper noun because the same domain element appeared in two different word forms. We assumed there is only one word form per domain element. Finally, there were 10 cases where an extra word was added or removed.

Problem	**Count**
"the one"	15
Wrong use of copular verb	6
Badly structured subordinate clause	6
Passive sentence	1
Possessive pronoun	1
Badly structured noun phrase	1
Rational numbers	3
Superlative	1
"Of the two ..."	1
Badly typed verb	14
Badly typed noun phrase	5
More than one word form for a domain element	1
Redundant word	7
Missing word (ellipse)	3

TABLE II: An overview of the different modifications

We refer to the full master thesis for a detailed overview of all the modifications. However, most of the modifications are small and they were always necessary to fit them in the grammar. No modifications were necessary to avoid a parse that resulted in a wrong translation.

IX. Related work

In this section we discuss some related work. We begin with three CNL's, two for knowledge representation and one with types. Finally, we discuss another framework that is based on λ-calculus.

Two of the most advanced controlled natural languages that can be used for knowledge representation are Attempto Controlled English (ACE) and Processable English (PENG). ACE [7] is a general purpose CNL. The language contains a large built-in vocabularium. This has as advantage that the user doesn't have to provide the vocabularium. However, for specifications in small domains, we usually do want to provide the vocabularium to make sure the specification only talks about the modelled domain. PENG [8] does ask the user to

provide its own content words. Moreover, there is a tool named ECOLE which gives suggestions while writing PENG sentences, namely about which linguistic constructs can follow the words already entered. This makes it easier to write correct PENG sentences. These languages mainly show that we can translate English into logic, but the literature on both these CNL's lacks a description of this translation process. Therefore, extending these CNL's is hard.

Another important CNL is RuleCNL [9]. It is a CNL that translates (a subset of) English into business rules. It's notable because it is the only CNL we could find that uses types. Interestingly, the paper lacks a description of the type system that was used and of the inferences that are supported. However figure 6 from [9] indicates that type checking of business rules in RuleCNL is supported.

Finally, Baral et al. [10], [11], [12] researched translating natural language into ASP programs. They do this based on λ-calculus and a Combinatorial Categorial Grammar (CCG). Each word gets one λ-ASP-expression per CCG category it can represent. The CCG then describes how to combine these expressions via λ-application. In their latest paper on this subject [12], Baral et al. explain how to learn both these λ-ASP-expressions and a probabilistic CCG grammar from a set of logic grid puzzles. They used a supervised machine learning method for this goal. Based on these methods they can solve unseen puzzles (without adapting the clues of the puzzles). They do not support other kind of inferences.

X. Conclusion

The framework of Blackburn and Bos can be used to translate a CNL into logic. This way, we assign semantics to such a CNL. Therefore, we can use such a CNL as a knowledge representation language within a knowledge base system. This allows different kinds of inference on the (constructed) natural language. For a logic grid puzzle, this can mean automatically solving the puzzle but also helping the user by giving hints or helping the author by giving the possible solutions.

Moreover, we expanded the framework with types. This allows us to translate the constructed CNL to a typed logic. We have proven that type inference is possible as well. More research into a typed CNL is definitely necessary. For example, what the value of a more complex type system is. Or if we can construct a CNL with a type system with multiple types per word. The type system can then derive the correct instance of the verb. E.g. it can differentiate between ordering drinks and ordering food.

References

[1] B. de Cat, B. Bogaerts, M. Bruynooghe, and M. Denecker, "Predicate logic as a modelling language: The IDP system," *CoRR*, vol. abs/1401.6312, 2014. [Online]. Available: http://arxiv.org/abs/1401.6312

[2] T. Kuhn, "A Survey and Classification of Controlled Natural Languages," *Computational Linguistics*, vol. 40, no. 1, pp. 121–170, mar 2014. [Online]. Available: http://www.mitpressjournals.org/doi/abs/10.1162/COLI_a_00168

[3] P. Blackburn and J. Bos, "Representation and inference for natural language," 2005. [Online]. Available: http://www.coli.uni-saarland.de/publikationen/softcopies/Blackburn:1997:RIN.pdf

[4] ——, "Working with discourse representation theory," *An Advanced Course in Computational Semantics*, 2006.

[5] Wikipedia, "Zebra Puzzle — Wikipedia, the free encyclopedia," http://en.wikipedia.org/w/index.php?title=Zebra%20Puzzleoldid=779870856, 2017, [Online; accessed 24-May-2017].

[6] S. Ryder, *Puzzle Baron's logic puzzles*. Indianapolis, Indiana: Alpha Books, 2016.

[7] N. E. Fuchs, K. Kaljurand, and T. Kuhn, "Attempto controlled english for knowledge representation," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5224 LNCS, pp. 104–124, 2008.

[8] R. Schwitter, "English as a Formal Specification Language," *Interpreting*, pp. 228–232, 2002. [Online]. Available: http://web.science.mq.edu.au/ rolfs/papers/n-lis2002.pdf

[9] P. B. F. Njonko, S. Cardey, P. Greenfield, and W. El Abed, "RuleCNL: A controlled natural language for business rule specifications," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8625 LNAI, pp. 66–77, 2014.

[10] C. Baral, J. Dzifcak, and T. C. Son, "Using Answer Set Programming and Lambda Calculus to Characterize Natural Language Sentences with Normatives and Exceptions," *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI-08)*, pp. 818–823, 2008.

[11] C. Baral, M. A. Gonzalez, and A. Gottesman, "The inverse lambda calculus algorithm for typed first order logic lambda calculus and its application to translating english to FOL," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7265, pp. 40–56, 2012.

[12] C. Baral and J. Dzifcak, "Solving Puzzles Described in English by Automated Translation to Answer Set Programming and Learning How to Do that Translation," *Knowledge Representation and Reasoning (KR)*, pp. 573–577, 2012.

Automatisch vertalen van logigrammen naar logica

KATHOLIEKE UNIVERSITEIT LEUVEN

FACULTEIT INGENIEURSWETENSCHAPPEN

Master Computer-wetenschappen

Masterproef
Jens Claes

Promotor
Marc Denecker

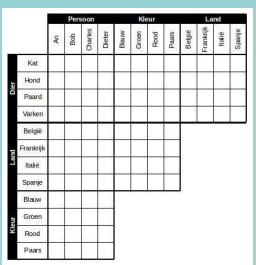
Begeleiders
Bart Bogaerts
Laurent Janssens

Academiejaar 2016-2017

Motivatie

In het *Knowledge Base*-paradigma wordt een probleem gereduceerd tot een specificatie waarop verschillende inferenties worden uitgevoerd. Een formele specificatie is echter moeilijk te lezen en te schrijven. Het automatisch vertalen van natuurlijke taal naar logica lost dit probleem op.

Deze thesis onderzoekt het vertalen van een logigram (een puzzel met een aantal constraints in natuurlijke taal) naar een formele specificatie in logica



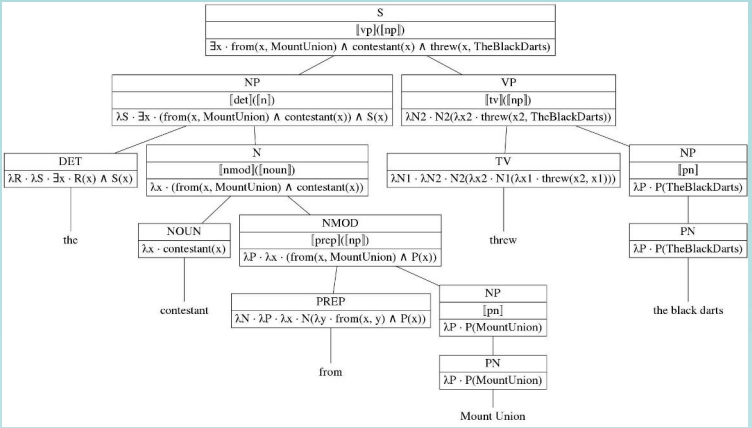
Voorbeeld - The contestant from Mount Union threw the black darts

Lexicon

Woord	Categorie	Betekenis
the	Lidwoord (DET)	$\lambda R \cdot \lambda S \cdot \exists x \cdot R(x) \wedge S(x)$
contestant	Substantief (NOUN)	$\lambda x \cdot contestant(x)$
from	Voorzetsel (PREP)	$\lambda N \cdot \lambda P \cdot \lambda x \cdot \mathrm{N}(\lambda y \cdot from(x,y) \wedge P(x))$
Mount Union	Eigennaam (PN)	$\lambda P \cdot P(MountUnion)$
threw	Overgankelijk werkwoord (TV)	$\lambda N1 \cdot \lambda N2 \cdot \mathrm{N2}(\lambda x2 \cdot N1(\lambda x1 \cdot threw(x2,x1)))$
the black darts	Eigennaam (PN)	$\lambda P \cdot P(TheBlackDarts)$

Grammatica

Grammaticale regel	Betekenis
S → NP VP	$[\![VP]\!]([\![NP]\!])$
NP → DET N	$[\![DET]\!]([\![N]\!])$
NP → PN	$[\![PN]\!]$
N → NOUN NMOD	$[\![NMOD]\!]([\![NOUN]\!])$
NMOD → PREP NP	$[\![PREP]\!]([\![NP]\!])$
VP → TV NP	$[\![TV]\!]([\![NP]\!])$



Een semantisch framework

(Blackburn en Bos 2005, 2006)

Het lexicon is verschillend per logigram. De grammatica is dezelfde voor alle logigrammen.

De betekenis van een woord is een functie van de lexicale categorie.

Compositionaliteit: de betekenis van een woordgroep is een combinatie van de betekenissen van de woorden waaruit ze bestaat. Zo wordt de betekenis van de woorden naar boven toe gepropageerd in de parse tree

Types en het formeel vocabularium

Veronderstelling: **elk woord** heeft **1 type** per logigram.

Bij meerdere constraints **unificeren** de woorden die meerdere keren voorkomen de types. Verdere unificatie verloopt via vragen aan de gebruiker i.v.m. synonymie van woorden.

Substantieven en eigennamen introduceren een *basistype*

Overgankelijke werkwoorden en voorzetsels introduceren een afgeleid *tuple-type* (met 2 basistypes als argument).

Eigennamen worden vertaald naar constanten van constructed types. Door unificatie van de basistypes worden deze eigennamen gegroepeerd.

Voorzetsels en overgankelijke werkwoorden introduceren een predicaat.

Extra axioma's (toegevoegd aan de theorie, specifiek voor logigrammen)

- Linken van predicaten a.d.h.v. hun signatuur
  - Bv. Twee predicaten met dezelfde signatuur zijn gelijk
- Elk predicaat is een bijectie
- Symmetrie-brekende axioma's

Resultaten

Gegeven: Aantal (basis)types, de constraints (in het Engels) en het logigram-specifiek lexicon

Extra vragen aan de gebruiker:

- Unificatie types (op basis van synonymie van woorden)
- Domein voor numerieke types

Resultaat: Vocabularium + Theorie in IDP

Experiment: Grammatica op basis van 10 puzzels

Evaluatie: 10 nieuwe puzzels allemaal vertaalbaar mits 80 (kleine) correcties aan de constraints in natuurlijke taal.

Conclusie: Succes mits beperking op gebruikte grammatica

Probleem	Aantal	Voorbeeld
Ontbrekende woorden	6	John's trip will begin before Janice's trip
Overtollige woorden	7	Wolfenden was said to be haunted by
> 1 type voor 1 woord	21	The trip starts begins at 9 and starts at Kiev
The one	15	… before the one tour starting …
Herschrijving NP	18	The comet Parks discovered by Parks
Type conversie	3	John finished before the man acting as doctor
Andere	10	$5.99 $6

P. Blackburn and J. Bos. Representation and inference for natural language. A first course in computational semantics. CSLI, 2005.

P. Blackburn and J. Bos. Working with discourse representation theory. An Advanced Course in Computational Semantics, 2006.

DTAI
Declaratieve Talen en Artificiële Intelligentie