VERKENNING VAN EEN METHODE OM REGELSPRAAK FORMEEL TE SPECIFICEREN_

Bas Hofmans, MSc.

21 juni 2022

HAN_UNIVERSITY
OF APPLIED SCIENCES

INHOUDSOPGAVE

INLEIDING

Dit document bevat de resultaten van een verkennend onderzoek naar het maken van een specificatie van RegelSpraak. RegelSpraak is ontstaan in het project complexiteitsreductie fase 2 omstreeks 2008. Het is gebaseerd op RuleSpeak en vanuit een pragmatisch perspectief ontwikkeld. Momenteel wordt RegelSpraak (inclusief GegevensSpraak en TestSpraak) in diverse projecten gebruikt als taal voor het specificeren van regels. In toenemende mate worden delen van fiscale wet- en regelgeving en beleid in RegelSpraak uitgedrukt.

RegelSpraak maakt gebruik van regelpatronen die bepalen hoe regels gestructureerd kunnen worden, hiermee bepaalt de verzameling regelpatronen welke regels met Regelspraak gespecificeerd kunnen worden, en hoe. RegelSpraak is op dit moment geïmplementeerd in ALEF en (gedeeltelijk) in RuleXpress, maar er is geen leidend, implementatie (tool)-onafhankelijk brondocument dat een standaard voor RegelSpraak definieert en dat als naslagwerk kan dienen voor implementerende partijen en/of tool-ontwikkelaars.

De Belastingdienst wil RegelSpraak onafhankelijk maken van de bestaande implementaties en een standaard voor RegelSpraak vaststellen. Binnen dit project wordt onderzocht hoe een dergelijke standaard gemaakt kan worden door een specificatiemethode voor de taal te ontwikkelen met het doel een eenduidige en begrijpelijk omschrijving te geven wat de taal en de taalelementen er in betekenen.

Om dit te bereiken is het nodig om voor de taal twee dingen te specificeren, namelijk de syntax van de taal waarmee de vorm van elementen gespecificeerd wordt en de semantiek van de taal waarmee betekenis aan de elementen gegeven wordt. Door deze twee aspecten te specificeren wordt het duidelijk hoe we aan kunnen geven welke vormen bepaalde taalelementen aan kunnen nemen en daarbij ook welke betekenis het heeft als deze taalelementen in het uitvoeren van een regel voorkomen in een bepaalde vorm.

Link met opleidingsproject

De voorbeelden waarmee aspecten van de specificatie in dit document geïllustreerd worden zijn afkomstig uit het opleiding_toka project. Ditzelfde project wordt gebruik in het Regelspraakdocument dat verdere uitleg over de taal geeft om daar ook voorbeelden te leveren.

Opbouw

In dit document bespreken we eerst de drie onderdelen van de specificatievorm die voor RegelSpraak gebruikt kan worden. Hierin bespreken we eerst de vorm waarmee de syntax gespecificeerd wordt, dan de conceptvorm en hoe taalelementen van de syntax naar de conceptvorm vertaald kan worden, en daarna de vorm waarmee de semantiek gespecificeerd wordt. Hierna geven we een aantal voorbeel

van specificaties van concepten. Waarna we een methode uitleggen waarmee een specificatie van een concept opgesteld kan worden, gevolgd door conclusies over het onderzoek. In de bijlage wordt nog toegelicht op welke manier deze specificatiemthode relateerd aan ALEF.

1 SYNTAXSPECIFICATIE:

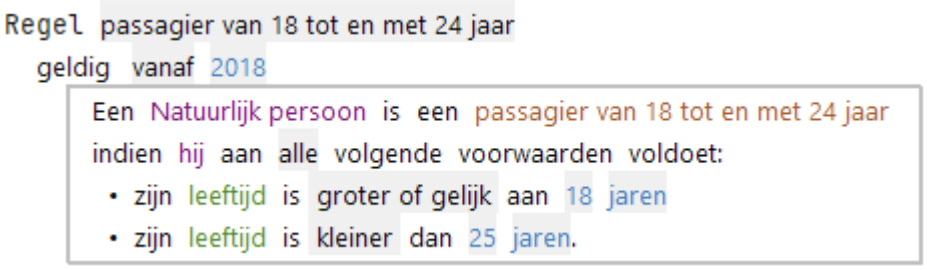
De syntax van Regelspraak wordt gespecificeerd als een grammatica volgens de EBNF-standaard met een aantal toevoegingen om alle structuren die in Regelspraak gebruikt worden te faciliteren. De syntax van RegelSpraak wordt gespecificeerd door voor alle taalelementen uit Regelspraak productieregels te specificeren die aangeven tot welke 'taal' deze elementen binnen regels kunnen leiden.

1.1 EBNF:

Het uitgangspunt van de grammaticaspecificatietaal is Extended Backus-Naur Form (EBNF). Er zijn veel verschillende precieze standaarden voor EBNF, hier gebruiken we als uitgangspunt de EBNF notatie zoals deze gebruikt wordt door de W3C (zoals omschreven in https://www.w3.org/Notation.html en https://www.w3.org/2001/06/blindfold/grammar).
Een syntaxspecificatie op basis van een dergelijke grammatica geeft de tekst die binnen de taal gegenereerd kan worden. De grammatica bestaat uit twee belangrijke componenten: de non-terminals en de bijbehorende productieregels. Non-terminals zijn de concepten uit de taal, waarvoor met productieregels wordt gespecificeerd welke verschillende vormen deze concepten aan kunnen nemen, hier kunnen ook weer nieuwe non-terminals in voorkomen. Als er in een concept ook weer andere concepten voor kunnen komen dan zullen deze als nieuwe non-terminals terugkomen in de productieregels.

1.2 Bijzonderheden:

In de grammatica gebruiken we drie speciale terminal tekens, namelijk de ¶ en het paar van --> en <--. Het ¶ teken geeft in de grammatica een punt aan waarin de tekst na dit teken op een nieuwe regel moet komen. Het paar van --> en <-- geven punten aan waar het niveau van indentatie vergroot en later weer verkleind wordt om hiermee geïndenteerde blokken aan te geven. Hierbij gaat het bijvoorbeeld om een geïndenteerd blok subcondities steeds op nieuwe regels.



Figuur 1: Regel met geïndenteerd blok condities, ook het regelblok is duidelijk geïndenteerd – Voorbeeldregel afkomstig uit de Kenmerktoekenning Natuurlijk Persoon regelgroep van het opleiding_toka project

1.3 Voorbeeld van syntax-specificatie:

Als voorbeeld hebben we hier de grammaticaregels van een aantal van de taalelementen uit Regelspraak:

```
Regel ::= "Regel " Name ¶ --> (RegelVersie ¶)* <--
RegelVersie ::= "geldig " GeldigheidsPeriode ¶ --> ActieIndienVoorwaarde<--
ActieIndienVoorwaarde ::= Actie (¶ "indien" Conditie)?
Conditie ::= EnkeleVoorwaarde | RegelVersieConditie |
SamengesteldeVoorwaarde | Uniciteit
SamengesteldeVoorwaarde ::= SamengesteldPredicaat
SamengesteldPredicaat ::= "er aan" Quantificatie "volgende voorwaarden wordt
      voldaan:" ¶ --> (Subconditie ¶)+ <--
Quantificatie ::= Alle | Geen | AantalQuantificatie
Alle ::= "alle"
Geen ::= "geen"
AantalQuantificatie ::= AantalQuantificatieConditie integer
AantalQuantificatieConditie ::= "ten minste" | "ten hoogste" | "precies"
Subconditie ::= Conditie
```

2 "TUSSENVORM": CONCEPTVORM:

Vanuit de specificatie van de syntax maken we voor Regelspraak niet direct een specificatie van de semantiek. De syntaxspecificaties geven duidelijk de mogelijke vormen van regels weer, maar bevatten ook veel natuurlijke taal die voor de semantiek geen informatie toevoegen en wel veel onduidelijkheid toevoegen in de specificatie van semantiekregels. Hiernaast zijn er in de vorm vaak veel taalvariaties mogelijk die uiteindelijk geen invloed hebben op hoe het concept daadwerkelijk 'ingevuld' wordt.
Hierdoor hebben we ervoor gekozen om een tussenvorm toe te voegen tussen de syntax en de semantiek, in deze vorm is de 'taligheid' uit de syntax gehaald, en worden taalelementen met meer structuur weergegeven. Hierin wordt een taalelement gerepresenteerd door de naam van het element samen met de andere taalelementen die eventueel nog in het element zitten.
Hiermee sluit deze tussenvorm ook nauwer aan op de manier waarop taalelementen als concepten binnen ALEF gerepresenteerd worden, daarom noemen we deze tussenvorm ook de conceptvorm.
Door de introductie van deze vorm is het ook nodig extra regels te specificeren die vastleggen hoe deze concepten van de syntaxvorm naar deze tussenvorm vertaald worden.

Concept:

- Parameter 1: type/waarde
- ……

In de specificatie van de vorm van een concept worden de types van de parameters aangegeven, als het gaat om instanties van een concept of de specificatie van regels waar de waarde van een parameter gebruikt wordt gaat het om de waarde van een parameter.

Voorbeeld ActieIndienVoorwaarde:

De conceptvorm van ActieIndienVoorwaarde wordt dan:

ActieIndienVoorwaarde:

- actie: Actie
- conditie: Conditie

Hiervoor kunnen we dan ook de regels specificeren waarmee de verschillende vormen die een ActieIndienVoorwaarde uit de syntax kan hebben naar deze vorm vertaald kunnen worden. Hiervoor zijn twee regels nodig, namelijk voor instanties van ActieIndienVoorwaarde waar er wel een conditie is en voor instanties waar er geen conditie is.

De twee regels, voor regels met daarin Actie a1 en Conditie c1, zijn dan als volgt:

`ActieIndienVoorwaarde: "a1"`

↓

ActieIndienVoorwaarde:

- Actie: a1
- Conditie: *Geen*

`ActieIndienVoorwaarde: "a1 ¶ indien c1"`

↓

ActieIndienVoorwaarde:

- Actie: a1
- Conditie: c1

3 SEMANTIEKSPECIFICATIE:

Met een specificatie van de semantiek beschrijven we wat er daadwerkelijk gebeurt als de regels van Regelspraak uitgevoerd worden. Dit doen we op basis van de conceptvorm, voor alle concepten beschrijven we voor de conceptvorm welk effect het heeft op de input als het uitgevoerd wordt. Om dit te doen gebruiken we operationele semantiek, specifiek de big-step variant die ook als natural semantics bekend staat. Op de basistheorie hiervan maken we wel een aantal aanpassingen om met alle functionaliteiten van Regelspraak om te kunnen gaan en om dit in de uitwerking een duidelijkere weergave te geven voor alle elementen.

3.1 Natural Semantics:

Voor de specificatie van de semantiek baseren we onze techniek op de natural semantics. In het bijzonder baseren wij ons op de beschrijving van natural semantics zoals die door Nielson & Nielson omschreven wordt in het boek Semantics with Applications: A formal introduction[1]. Met deze natural

[1] http://www.cs.ru.nl/~herman/onderwijs/semantics2019/wiley.pdf

semantics wordt voor elk taalelement met een of meerdere afleidingsregels omschreven wat het effect is van het uitvoeren van dit taalelement in een bepaalde toestand.
De taalelementen waar we dit voor specificeren zijn de taalelementen in conceptvorm zoals we die in de vorige stap opgesteld hebben. Voor een concept met eventueel een aantal onderliggende concepten omschrijven we wat er gebeurt als deze in een toestand uitgevoerd wordt. De toestand waarin regels uitgevoerd worden is in ons geval de data die als input gegeven wordt en waar de regels op toegepast worden. De toestand is het middel waarmee wijzigingen toegepast worden en waarmee dus de daadwerkelijk functionaliteit van de taalelementen bepaald wordt. Voor Regelspraak betekent dit dat wijzigingen of toevoegingen aan de data het middel zijn waarmee het effect van Regels en taalelementen bepaald wordt.
De semantiek voor een taalelement wordt omschreven door middel van een of meerdere regels. Meerdere regels worden gebruikt om verschillende invullingen van subconcepten en waardes in de data af te vangen. Door verschillende regels op te stellen kunnen hiermee alle mogelijkheden voor subconcepten en waardes afgedekt worden en kan daarmee een volledige semantiek voor het concept opgesteld worden. Dit wil niet zeggen dat een enkele regel alleen voor een specifieke waarde van subconcepten geldig is, binnen een regel kan ook binnen subconcepten en de evaluatie daarvan gebruik gemaakt worden van variabele of arbitraire elementen en als deze variabel op dezelfde manier afgehandeld kunnen worden. Als verschillende uitkomsten of waardes echter leiden tot structureel andere conclusies is het nodig hier meerdere regels voor te maken.
Regels bestaan uit twee delen: de conclusie en eventuele voorwaarden. De conclusie beschrijft het effect van het uitvoeren van het hele concept, met een dataobject als invoertoestand en met een potentieel anders dataobject als uitvoertoestand. De voorwaarden beschrijven het welke uitkomst het uitvoeren of evalueren van subconcepten moet hebben om te zorgen de conclusie geldig is.

3.2 Vorm:

In de notatievorm van regels wijken we af van de techniek die in Nielson en Nielson gebruikt wordt. De notatie zoals die in het boek gebruikt wordt, leent zich niet zo goed voor het werken met objecten in een vorm als de conceptvorm die wij hier gebruiken. Hierdoor werken wij niet met een deelstreep om regels mee te specificeren maar gebruiken wij hiervoor een tabel waarin de conclusie en de voorwaarden van de regels vastgelegd worden.

Rule: Naam		
Conclusie:		
Concept:	**Begintoestand:**	**Eindtoestand:**
- subconcepttype: a1	dataobject	dataobject’
- …		
Voorwaarden:		
Subconcepttype: a1	dataobject	dataobject’
…		

In de tabel staat eerst de naam van de regel, vaak gebaseerd op het concept waarvoor de regel is. Hieronder staat de conclusie: links het concept waar de regel op van toepassing is met daaronder in een geïndenteerde lijst de subconcepten die er in het concept zitten. Bij ieder subconcepttype staat ook een naam aangegeven, dit kan ofwel een naam van een variabele zijn ofwel een constante. In het bovenstaande voorbeeld is de naam van het subconcept ‘a1’, een variabele naam. Bij een variabele weten we niets over het object behalve dat het het type van het subconcept heeft, hierdoor zal er vaak in de voorwaarde nog een afleiding van dit subconcept nodig zijn om de betekenis hiervan te achterhalen. De naam kan ook een constante waarde van het subconcepttype zijn, in dit geval is de regel alleen van toepassingen voor instantie van het concept waarbij het subconcept gelijk is aan de aangegeven constante, en zullen andere regels nodig zijn om variaties hierop te omsluiten. Rechts van het concept staat de begintoestand, dit is ook te zien als de input in het concept. Dit is de toestand van de inputdata voordat het concept uitgevoerd is. Weer rechts hiervan staat de eindtoestand ofwel de output van het uitvoeren van het concept.

Hieronder staan de voorwaarden: per regel een subconcept waarvoor het uitvoeren vanuit een begintoestand uitgevoerd wordt op een bepaalde uitkomst moet uitkomen om ervoor te zorgen dat de conclusie klopt. Hiermee zijn het de voorwaarden die moeten gelden om de uitkomst in de conclusie waar te maken. In de meeste gevallen zullen de voorwaarden op volgorde staan van hoe de subconcepten voorkomen in het hoofdconcept, er zijn echter ook situaties waarin het resultaat uit de evaluatie van een bepaald subconcept nodig is bij het evalueren van een ander subconcept. In zulke gevallen komen de voorwaarden waarvan de uitkomst nodig is om een andere voorwaarde te evalueren eerder staan dan de voorwaarden waarvoor deze nodig zijn. Als een bepaald subconcept in een specifieke regel niet daadwerkelijk van belang is voor de uitkomst van de regel hoeft hier ook geen voorwaarde voor gemaakt te worden.

Naast voorwaarden waarin een subconcept geëvalueerd wordt, kunnen er ook voorwaarden voorkomen die niet de waarde of status van een subconcept evalueren maar waarbij specifieke eisen gecheckt worden, zoals dat een attribuut een bepaalde waarde heeft, of dat een kenmerk geldt voor een object. Zulke voorwaarden zullen onderaan de lijst voorwaarden voorkomen, deze maken namelijk vaak ook gebruik van de resultaten van het evalueren van andere voorwaarden. Hiermee kunnen eisen aan een bepaalde afleidingsregel gesteld worden.

De referentie naar `dataobject` dat in de tabellen voorkomt gaat om een arbitrair dataobject, waar we in de regelspecificatie misschien nog weinig vanaf weten, de andere dataobjectreferentie naar `dataobject'` gaat om een ander arbitrair data object. Deze refereren wel naar hetzelfde object in de conclusie en in de voorwaarden, hierdoor is de interpretatie voor de vorm in dit geval dat ALS het

uitvoeren van het subconcept met als invoer `dataobject` leidt tot een uitvoer van `dataobject'`, DAN leidt ook het uitvoeren van het concept met als invoer `dataobject` tot uitvoer `dataobject'`.

We bekijken als voorbeeld ook nog een van de regels voor het concept ActieIndienVoorwaarde:

Rule: ActieIndienVoorwaarde$_{true}$		
Conclusie:		
ActieIndienVoorwaarde:	**Begintoestand:**	**Eindtoestand:**
- actie: a1	dataobject	dataobject'
- conditie: c1		
Voorwaarden:		
Actie: a1	dataobject	dataobject'
Conditie: c1	dataobject	true

Dit is een van de regel voor het concept ActieIndienVoorwaarde, specifiek geeft deze regel aan wat het gedrag is voor het geval dat de conditie als true evalueert. Hier zien we dat in het conclusie gedeelde het concept ActieIndienVoorwaarde gespecificeerd staat met de twee subconcepten, de actie en de conditie. Het wordt uitgevoerd op `dataobject` en heeft als uitvoer `dataobject'`. Hiervoor zijn twee voorwaarden: ten eerste moet het uitvoeren van de actie met als invoer dataobject tot diezelfde uitvoer `dataobject'` leiden. Dit geeft aan dat het daadwerkelijk aanpassen van de data gebeurt bij het uitvoeren van de actie, dit kan bijvoorbeeld een gelijkstelling zijn waarbij nieuwe waardes voor attributen aan de data toegevoegd worden, deze aanpassing komt om deze manier terug in het uitvoeren van de volledige ActieIndienVoorwaarde. De tweede voorwaarde is dat de conditie uitgevoerd met als invoer `dataobject` evalueert tot true, dus dat de conditie waar is. Is dit niet het geval dan is een van de andere regels voor ActieIndienVoorwaarde nodig, waarbij de actie niet uitgevoerd wordt. Het gebruik van deze vorm heeft wel een duidelijk nadeel ten opzichte van de vorm gebruikt door Nielson & Nielson: omdat de tabellen niet in elkaar te combineren zijn, is het niet mogelijk om duidelijke afleidingsbomen te maken die een afleiding van de semantiek van een volledige regel geeft. Dit is echter geen groot probleem voor het doel van de specificatie, het belangrijkste is om duidelijk de semantiek van elk concept te kunnen geven, en daar leent de gekozen vorm zich goed voor.

3.3 Bijzonderheden:

De standaardopzet voor natural semantics biedt een goed framework waarmee de semantiek van formele talen vastgelegd kan worden. Echter is de manier waarop het precies opgesteld is niet direct geschikt om voor Regelspraak te gebruiken. Ook moeten we een aantal functionaliteiten toevoegen om op alle functionaliteit in Regelspraak goed te kunnen vertegenwoordigen. Hierdoor hebben we een aantal aanpassingen en toevoegingen op de standaard natural semantics die we in deze paragraaf bespreken.

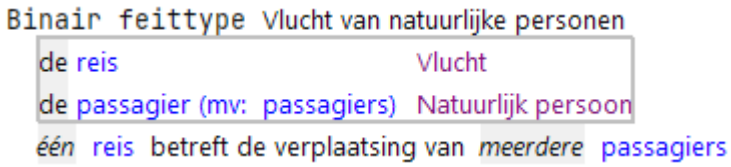
3.3.1 Gegevensmodel:

De regels van Regelspraak worden uitgevoerd binnen het kader van een gegevensmodel. In het gegevensmodel worden de objecttypes die in regels benaderd kunnen worden vastgelegd. Ook worden de attributen en kenmerken die deze objecttypes hebben vastgelegd, worden door middel van feittypes relaties tussen verschillende objecten aangegeven, en worden globale parameters gedefinieerd.

```
Objecttype de Natuurlijk persoon
    --- kenmerken
    de passagier jonger dan 18 jaar (mv: passagiers jonger dan 18 jaar) kenmerk;
    de passagier van 18 tot en met 24 jaar (mv: passagiers van 18 tot en met 24 jaar) kenmerk;
    de passagier van 25 tot en met 64 jaar (mv: passagiers van 25 tot en met 64 jaar) kenmerk;
    de passagier van 65 jaar of ouder (mv: passagiers van 65 jaar of ouder) kenmerk;
    het recht op duurzaamheidskorting kenmerk (bezittelijk);
    --- attributen
    het identificatienummer Numeriek (geheel getal);
    de geboortedatum Datum in dagen;
    de leeftijd Tijdsduur in jaren;
    de leeftijd numeriek Numeriek (geheel getal);
    het burgerservicenummer Tekst;
    de belasting op basis van afstand (mv: belastingen op basis van afstand) Bedrag;
    de belasting op basis van reisduur (mv: belastingen op basis van reisduur) Bedrag;
    de te betalen belasting Bedrag;
    de woonregio factor Numeriek (geheel getal);
    de treinmiles (mv: treinmiles) Numeriek (geheel getal);
    het maximaal aantal te ontvangen treinmiles Numeriek (geheel getal);
```

Figuur 2: Voorbeeld van het Objecttype Natuurlijke Persoon met attributen en kenmerken – Afkomstig uit het gegevensmodel van het opleiding_toka project

De objecttypes in het gegevensmodel geven de soorten objecten waar regels iets over kunnen zeggen, denk hierbij aan bijvoorbeeld ‘persoon’, ‘huis’, of ‘aanvraag’. Attributen en kenmerken worden aan deze objecttypes toegevoegd om informatie aan instanties van de objecttypes toe te voegen. Een voorbeeld van een attribuut is ‘leeftijd’ van het objecttype ‘Natuurlijk persoon’, zoals te zien in het bovenstaande voorbeeld. Kenmerken verschillen niet veel van attributen, echter zijn deze beperkt tot het booleaanse datatype, dit wil zeggen dat ze enkel ‘waar’ of ‘onwaar’ kunnen zijn. Kenmerken kunnen ook gebruikt worden om naar een bepaald deel van de instanties van een objecttype te refereren. Als er voor het onderwerp van een regel een kenmerk genoemd wordt in plaats van een type heeft de regel betrekking op alle objecten van het type dat bij het kenmerk hoort waarvoor het kenmerk true is.



Figuur 3 Voorbeeld van meervoudig feittype – Afkomstig uit het gegevensmodel van het opleiding_toka project

Door middel van feittypes worden relaties tussen objecttypes vastgelegd, hierin worden twee (mogelijk dezelfde) objecten met verschillende ‘rollen’ aan elkaar gerelateerd. Denk hierbij bijvoorbeeld aan objecten van de types ‘Natuurlijk persoon’ en ‘Vlucht’, die in een feittype met de rollen ‘passagier’ en ‘reis’ gekoppeld worden om aan te geven welke person(en) een bepaalde reis gemaakt hebben. Feittypes kunnen zowel enkelvoudig als meervoudig zijn, bij enkelvoudige feittypes wordt een object aan hooguit één ander object gekoppeld. Bij meervoudige feittypes kan een object aan meerdere objecten die de andere rol aannemen gekoppeld worden, het voorbeeld hierboven is een meervoudig feittype, omdat één reis gekoppeld kan worden aan meerdere passagiers.

- maal-expressie met attribuut en “harde” waarde:
  De te betalen belasting van een passagier moet berekend worden als zijn belasting op basis van afstand maal 25.

Figuur 4 Voorbeeld referentie naar onderwerp door middel van rol – deze voorbeeldregel is afkomstig uit het uitleggende Regelspraakdocument

Net als met kenmerken van objecten kan er ook met rollen uit een feittype naar een bepaalde subgroep van een objecttype gerefereerd worden. Zo kan er bijvoorbeeld gerefereerd worden naar de ‘te betalen belasting van een passagier’ zoals in de bovenstaande regel. Hierdoor is de regel van toepassing op alle objecten met het type ‘Natuurlijke persoon’ die als passagier aan een reis gekoppeld zijn door het feittype.

Voor de specificatie van de semantiek van Regelspraak bekijken we de feittypes in de data als wiskundige relaties, dit betekent dat een feittype een verzameling is van paren objecten die aan elkaar gekoppeld zijn. Deze verzamelingen bevatten de paren van objecten die volgens het feittype aan elkaar gerelateerd zijn, zoals een persoon en het huis waar deze persoon in woont.

```
// PARAMETERS
Parameter het PERCENTAGE REISDUUR EERSTE SCHIJF : Percentage (geheel getal)
Parameter het PERCENTAGE REISDUUR TWEEDE SCHIJF : Percentage (geheel getal)
Parameter het PERCENTAGE REISDUUR DERDE SCHIJF : Percentage (geheel getal)
Parameter de BOVENGRENS REISDUUR EERSTE SCHIJF : Numeriek (geheel getal)
Parameter de BOVENGRENS REISDUUR TWEEDE SCHIJF : Numeriek (geheel getal)
Parameter de BOVENGRENS AFSTAND EERSTE SCHIJF : Numeriek (geheel getal)
```

Figuur 5 Voorbeeld parameters – Afkomstig uit het gegevensmodel van het opleiding_toka project

Naast deze objecttypes en feittypes worden in het gegevensmodel ook globale parameters aangegeven. Deze parameters bevatten constante waarden die vanuit regels aangesproken kunnen worden. Binnen het gegevensmodel wordt voor de parameters de naam en het datatype vastgesteld, de waarde van de parameters is constant en er wordt binnen regels met de naam naar de parameters gerefereerd.

3.3.2 Data:

De regels van Regelspraak worden toegepast op data, de regels gebruiken informatie uit de data en passen de data aan om een effect te hebben. Bij het specificeren van de semantiek van regels en de taalelementen die daarin voorkomen is het daarom belangrijk dat de data meegegeven wordt. Hierdoor moeten we ook bepalen hoe we data binnen een regel interpreteren.

De regels van Regelspraak worden in de praktijk uitgevoerd op een verzameling van data. Om de semantiek van regels te specificeren kijken we echter naar het effect dat een regel heeft als deze toegepast wordt op een enkel dataobject uit de verzameling van data. Dit maakt het mogelijk om veel duidelijker aan te geven hoe data aan een object toegevoegd wordt of data van een object aangepast wordt door het toepassen van een regel. Door deze semantiek voor enkele dataobjecten dan toe te passen op alle dataobjecten uit de verzameling komt het effect op de hele dataverzameling naar voren. In de specificatie van de semantiek van taalelementen zal ‘dataobject’ voorkomen als de toestand waarin de regel uitgevoerd wordt. In deze specificaties refereert de term dataobject dan naar het enkele dataobject waarvoor we willen specificeren wat het effect van de regel is. Het specificeren van de afleidingsregels voor een taalelement doen we algemeen, dus we stellen de regels vast voor alle mogelijke vormen en inputdata die een element kan krijgen. Hierdoor werken we bij deze specificaties met een arbitrair dataelement, een element waar we nog niets vanaf weten. De semantiekregels moeten daardoor opgesteld worden voor de eigenschappen die de data potentieel kunnen hebben.

Regels hebben een effect doordat ze data gebruiken en veranderen. Hierdoor is het dus belangrijk om duidelijk te specificeren op welke manier data aangehaald wordt en op welke manier aanpassingen hierin genoteerd worden. Regels worden uitgevoerd op een dataobject uit de inputverzameling. De daadwerkelijke gegevens die aan dat dataobject gerelateerd zijn zitten in de attributen en kenmerken die dat dataobject heeft. In werkelijkheid zijn het dus ook deze attributen en kenmerken die aangepast moeten worden om de data aan te passen.

Dit betekent dus dat we moeten specificeren hoe we aangeven dat de waarde van een bepaald attribuut of kenmerk van een object verandert. Dit kunnen we vrij direct uit de natural semantics standaard van Nielson & Nielson halen. Hiervoor kunnen we het dataobject als wiskundige functie modelleren, die gegeven een bepaal attribuut of kenmerk de bijbehorende waarde teruggeeft. Hierdoor kunnen we dan functienotatie gebruiken om aan te geven dat de waarde van een bepaald attribuut of kenmerk veranderd is:

$$dataobject[attribuut \mapsto waarde]$$

Op deze manier wordt een object aangegeven waarbij een attribuut een bepaalde waarde gegeven wordt, het geheel is dan op zichzelf weer een aangepast dataobject en kan op dezelfde manier weer gebruikt worden om waardes op te vragen of om verdere aanpassingen te doen. Om te zien hoe dit daadwerkelijk in regels gebeurt, kijken we naar de definitie van de semantiek van het gelijkstelling taalelement:

Rule: Gelijkstelling		
Conclusie:		
Gelijkstelling:	**Begintoestand:**	**Eindtoestand:**
- Selectie: links	dataobject	Dataobject[selectie ↦ waarde]
- Expressie: rechts		
Voorwaarden:		
Selectie: links	dataobject	selectie
Expressie: rechts	dataobject	waarde

In de regel zien we op welke manier de gelijkstelling het dataobject daadwerkelijk aanpast. De selectie is een attribuut of kenmerk van het dataobject waar data van veranderd wordt. De nieuwe waarde die aan deze selectie toegewezen moet worden, wordt verkregen door de expressie hiervan te evalueren gegeven de huidige status van het dataobject. In de eindtoestand zal het attribuut of kenmerk waar de selectie naar refereert de waarde waar de expressie naar evalueert hebben.

Anders verwoord geldt de conclusie dat na het uitvoeren van de gelijkstelling het attribuut dan wel kenmerk selectie van het object dataobject naar ‘waarde’ verwijst als de expressie rechts gegeven het huidige dataobject tot ‘waarde’ evalueert.

3.3.3 Onderwerpreferenties:

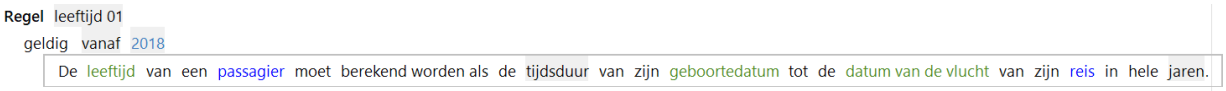
De onderwerpketen speelt een belangrijke rol in de functionaliteit van Regelspraak, de onderwerpketen maakt het mogelijk om binnen regels door het gegevensmodel te navigeren. Om alle referenties naar het gegevensmodel goed te kunnen verwerken zijn er twee aspecten nodig: ten eerste moet er een kader zijn waarbinnen referenties naar het gegevensmodel een semantiek krijgen en formeel interpretabel worden, en ten tweede moeten we specificeren hoe er genavigeerd kan worden tussen de verschillende elementen van het gegevensmodel.

Met betrekking tot de benadering van onderwerpreferenties binnen regels gaat het ook om de benadering van het dataobject waar we de regel op toepassen. Het dataobject waar de regel op toegepast wordt zien we in de regels als de begintoestand waarop veranderingen aangepast worden. Zoals eerder gespecificeerd hebben we het hierbij over een enkel dataobject. Hierbij is de term dataobject alleen een referentie naar het object, de daadwerkelijk data staat in de attributen en kenmerken die erbij horen. Hierbij bekijken we deze datareferentie dan ook wel in de context van de volledige dataverzameling, wat simpelweg betekent dat we via het object met rollen kunnen navigeren naar gerelateerde objecten in de dataverzameling.
Hierbij gaat het ook om relatieve referenties naar een onderwerp, zoals in de volgende voorbeeldregel:

- maal-expressie met attribuut en “harde” waarde:
  De te betalen belasting van een passagier moet berekend worden als zijn belasting op basis van afstand maal 25.

Figuur 6 Voorbeeld regel met relatieve onderwerpsreferentie – deze voorbeeldregel is afkomstig uit het uitleggende Regelspraakdocument

In deze regel wordt met de term ‘zijn’ gerefereerd naar het onderwerp ‘passagier’ dat in een ander deel van de regel vastgelegd wordt. Voor een werkende specificatie van de semantiek moet het in de afleiding van de betekenis van de conditie in deze regel, waar ‘zijn’ in voorkomt, duidelijk zijn waar dit naar refereert. Door onze benadering van de data vergt dit minder extra aandacht, doordat we ons richten op de uitvoering van een regel op een enkel dataobject en al controleren of het type van het object aansluit bij het onderwerp van de regel, weten we dat een relatieve referentie naar het onderwerp altijd naar het dataobject zal verwijzen.
Ten tweede moeten we kijken naar het navigeren tussen verschillende elementen in het gegevensmodel. Binnen regels wordt vanaf een bepaald onderwerp, een dataobject, naar andere dataobjecten genavigeerd door middel van de rollen uit de feittypes in het gegevensmodel. Door de rol los te noemen of deze verbonden aan het object te noemen kan hiernaa gerefereerd worden.

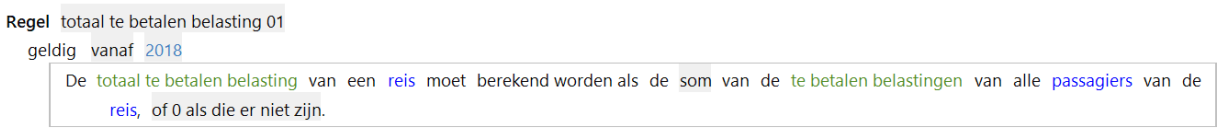


Figuur 7 Voorbeeld van regel met navigatie door middel van rol – deze voorbeeldregel is afkomstig uit de Regelgroep Leeftijd in het opleiding_toka project

In het bovenstaande voorbeeld komt dit voor, de regel heeft betrekking op alle objecten van het type Natuurlijk Persoon die als passagier deelnemen in het feittype 'Vlucht van natuurlijke personen'. Om de leeftijd van de passagier te berekenen is informatie nodig uit de reis waarin deze persoon deelneemt als passagier. Hiervoor wordt door te refereren naar 'zijn reis' het object van het type Vlucht opgehaald waaraan de passigier door het feittype gekoppeld is. Van deze reis wordt data uit een attribuut van deze reis gebruikt om de leeftijd van de passagier te bepalen.

Om duidelijk te hebben wat het nu betekent als een rol gebruikt wordt om van een object naar een nieuw object te navigeren moeten we ook dit nog specificeren. Dit doen we door de rollen als wiskundige functies te definiëren, deze functies geven voor de rol en het object waar het op toegepast wordt het object of de objecten terug waaraan het object gerelateerd is, of 'geen' als deze niet bestaat of bestaan.

Hierbij zit er een verschil tussen rollen uit enkelvoudige feittypes en rollen uit meervoudige feittypes. Als een rol uit een enkelvoudig feittype komt is er maximaal één object aan het onderwerp gekoppeld, het resultaat van de rol is dus altijd ofwel een enkel object, of 'geen'. Als een rol uit een meervoudig feittype komt kan het zo zijn dat hier meerdere objecten uit terug komen, hierdoor komt er niet een object terug, maar een verzameling objecten, ook als er in de werkelijkheid maar één object aan het onderwerp gekoppeld is komt er een verzameling objecten terug. Het resultaat van de rol toegepast op het onderwerp is dus ofwel een verzameling van objecten, of 'geen'. Als er een verzameling ontstaat door het aanroepen van een rol op een object gaat dit vaak gepaard met een som of aantal operator om de data die voor de objecten in de verzameling opgehaald wordt uit attributen terug te brengen naar een enkel getal of waarde omdat dit vaak nodig is voor de verdere verwerking in de regel.



Figuur 8 Voorbeeld van regel waarin het resultaat van een rol een verzameling objecten is – deze voorbeeldregel is afkomstig uit de Vluchtinformatie regelgroep in het opleiding-toka project

In de bovenstaande regel komt een rol voor uit een meervoudig feittype. Vanuit een reis wordt gerefereerd naar alle passagiers die aan die reis verbonden zijn, dit kan dus ofwel een verzameling opleveren, of 'geen'. In het geval dat er een verzameling natuurlijke personen als resultaat is wordt voor elk van deze natuurlijke personen het attribuut 'te betalen belastingen' uit het dataobject opgevraagd, hierover wordt dan de som berekend, omdat de waarde die uiteindelijk toegewezen wordt in de regel

een enkele waarde is. Ook wordt het geval afgevangen dat er geen passagiers aan de reis gekoppeld zijn en het aanroepen van de rol dus geen ‘oplevert’, in dat geval wordt er een constante waarde toegewezen.

Ook kunnen rollen geschakeld worden en op elkaar aangeroepen worden, hierbij blijft dan ook dezelfde interpretatie van de rollen gelden. Zolang dit om rollen uit enkelvoudige feittypes gaat wordt dit niet ingewikkelder, elke rol stuurt het onderwerp weer door naar het volgende object. Het kan hierbij echter ook gaan om het toepassen van een rol op een verzameling objecten uit een eerdere rol uit een meervoudig feittype. Als de nieuw toegepaste rol uit een enkelvoudig feittype komt, wordt deze rol simpelweg toegepast op alle objecten in de verzameling objecten, waardoor een verzameling ontstaat van de objecten die gekoppeld zijn aan alle objecten in de originele verzameling. Komt de nieuw toegepaste rol weer uit een meervoudig feittype dan wordt deze rol ook toegepast op alle elementen uit de verzameling objecten, omdat hier steeds een verzameling voor terug komt zal het resultaat hierbij dus een verzameling met daarin andere verzamelingen zijn.

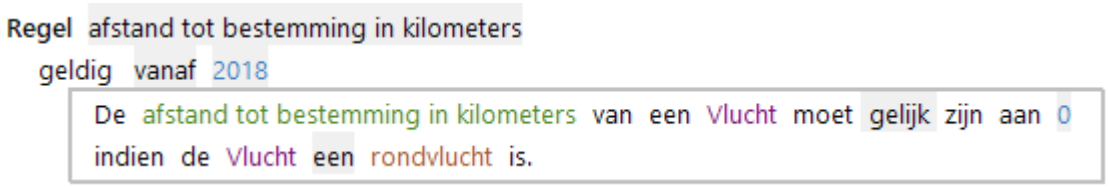
Ook attributen en kenmerken kunnen van resulteren objecten uit rolreferenties opgevraagd worden door de functies hiervan te gebruiken. Dit geldt ook als het om een verzameling objecten gaat uit een referentie naar een rol van een meervoudig feittype, hierbij wordt dan met een attribuut of kenmerk de waardes voor deze attributen voor alle objecten in de verzameling opgevraagd, wat daardoor een verzameling aan waardes van attributen oplevert.

3.3.4 Afleidingen voor condities, expressies, waardes:

Bij het specificeren van de semantiek van taalelementen van Regelspraak focussen we ons doorgaans op het omschrijven wat er gebeurt met het dataobject die als input voor de regel dient. Doordat het dataobject aangepast wordt heeft de regel namelijk uiteindelijk een bepaald effect. Voor bepaalde taalelementen moeten we echter een andere benadering gebruiken. Voor condities en expressies beschrijven we niet welke invloed deze hebben op het dataobject, maar de waarde waar ze naar evalueren. Condities zijn in een bepaalde toestand waar of niet waar, dus voor condities omschrijven we op welke manier deze naar een booleaanse waarde evalueren. Expressies evalueren in een bepaalde toestand tot een waarde, dus voor expressies omschrijven de regels naar welke waarde de expressie evalueert op basis van de gegeven toestand.

Voor beide wordt er geëvalueerd in een bepaalde toestand, waarbij de toestand weer het dataobject die we als invoer voor de regel gebruiken is. Dit invoerdataobject is niet voor alle soorten expressies en condities daadwerkelijk relevant. Als voorbeeld kan ook het getal ‘2’ als expressie gegeven worden, en die expressie zal ongeacht het invoerdataobject tot 2 evalueren. Toch gebruiken we ook voor elementen die de data niet in de evaluatie gebruiken een vorm met het dataobject als input. Dit is omdat deze elementen ook een instantie zijn van een algemener concept die potentieel wel het

dataobject gebruiken, en om daarvoor consistent in notatie te zijn en om het geen onnodige onduidelijkheid te laten ontstaan over het ‘verdwijnen’ van het dataobject.



Figuur 9 Voorbeeld van regel met numerieke literal – deze voorbeeldregel is afkomstig uit de regelgroep Consistentiecontrole afstand tot bestemming uit het opleiding_toka project

De bovenstaande regel gebruikt een dergelijk element waarvoor het invoerdataobject niet zo belangrijk is. De ‘0’ die in de regel voorkomt zal, ongeacht de andere omstandigheden, altijd tot de waarde 0 evalueren. In dit geval is dit een NumeriekeLiteral, een concept wat een numerieke waarde bevat (in het subconcept waarde) en deze in evaluatie in alle geval direct oplevert. De semantiekregel van dit concept is dus erg simpel:

Rule: NumeriekeLiteral		
Conclusie:		
NumeriekeLiteral:	**Begintoestand:**	**Eindtoestand:**
- NumeriekDataType: waarde	dataobject	waarde

De regel geeft simpelweg direct de waarde terug die in het subconcept staat. Hiervoor hoeft het dataobject niet geraadpleegd te worden. Het is namelijk toch van belang dat dit dataobject als invoer meegegeven wordt. Een NumeriekeLiteral is namelijk een instantie van een expressie. Daar waar een expressie verwacht wordt kan dit dus altijd een NumeriekeLiteral zijn. Omdat er bij het evalueren van een expressie wordt verwacht dat er een dataobject ingevoerd wordt, is het voor de consistentie goed om dit ook bij alle instanties van expressies te hebben, ook als dit dataobject niet in de evaluatie gebruikt wordt.

3.3.5 Functies voor verzamelingen:

Voor de specificatie van Regelspraak zijn een aantal speciale functies nodig, belangrijk hierbij zijn functies die het werken met verzamelingen en lijsten van onbepaalde lengte. Door het gebruik van rollen uit meervoudige feittypes kan het zijn dat er op momenten net een verzameling van meerdere objecten gewerkt wordt. Als er gerefereerd wordt naar ‘alle kinderen van’ een persoon kan het bijvoorbeeld het geval zijn dat dit meerdere objecten oplevert. Ook komen er in samengestelde condities lijsten van onbepaalde lengte voor waar statements over de waarheid gemaakt moeten kunnen worden, hiervoor zijn logische statements nodig.

Hiervoor zijn een aantal functies nodig, de eerste hiervan zijn functies om de hoeveelheid objecten te tellen en om de waardes van een optelbaar attribuut van al deze objecten op te tellen. Hiervoor introduceren we de count en sum functies, die deze twee resultaten geven. Voor de volledigheid geven beide functies de waarde 0 terug als de verzamelingen waar ze op aangeroepen worden leeg zijn.
Ook is het nodig om notitie en functies te introduceren om met lijsten van onbepaalde lengte te werken. Dit om te kunnen omgaan met lijsten en condities die in regels voor kunnen komen. Voor deze lijsten weten we van tevoren niet hoe lang de lijst met regels gaat zijn, dus hebben we een notatie nodig die ongeacht de lengte van de lijst regels een betekenis aan de regel kan geven. Hierbij is het vooral belangrijk dat we de resultaten van het evalueren van de verschillende condities in zulke lijsten kunnen combineren. Omdat condities tot ‘true’ of ‘false’ evalueren zijn hiervoor concepten uit de logica uitermate geschikt. Specifiek kunnen we goed gebruik maken van kwantoren uit de logica, die een onbepaalde hoeveelheid ‘true’ of ‘false’ waardes samenvoegen tot één waarde op basis van deze waardes. Omdat deze kwantoren in de semantiekregels voor moeten komen worden deze ook binnen de vorm van de semantiekregels gezet worden, hoe we dat doen tonen we hieronder bij de kwantoren. Hiernaast nemen we ook de negatie operator over, deze operator geeft voor een booleaanse waarde het tegenovergestelde, ‘false’ als de waarde ‘true’ is en vice-versa.
We gebruiken de volgende notatie voor deze nieuwe functionaliteiten:
$[s]$: Een verzameling van elementen van type s
$count\ [s]$: de count functie die de hoeveelheid elementen in de verzameling oplevert
$sum\ [s]$: de sum functie die de optelling van alle elementen in een verzameling van optelbare elementen oplevert
$\neg b$: de negatie van boolean b, als b is true, is de negatie hiervan false en vice-versa.

De notatie voor kwantoren is gebaseerd op het voorkomen van deze kwantoren in de voorwaarden van een regel, dus deze worden in een regel van een tabel weergegeven zoals deze in de voorwaarden van een semantiekregel voor zouden kunnen komen.

∀ object:[Object]	dataobject	boolean

De bovenstaande kwantor is de universele kwantor zoals deze in de voorwaarde van een regel voor kan komen, deze kwantor is universeel omdat deze stelt dat de eis voor alle objecten in een verzameling moet gelden. Aan de voorwaarde is dus alleen voldaan als het voor alle objecten in de verzameling zo is dat het afleiden van de booleaanse waarde van deze objecten gegeven de data leidt tot de aangegeven boolean waarde.

∃ object:[Object]	dataobject	boolean

De volgende kwantor is de existentie kwantor. Bij de existentie kwantor gaat het om het bestaan van een object waarvoor de voorwaarde geldt. Aan de voorwaarde wordt dus voldaan als voor minstens één van de objecten in de verzameling objecten geldt dat het afleiden van de booleaanse waarde van dit object gegeven de object tot de gespecificeerde boolean waarde leidt.

$\exists_n^m$ object:[Object]	dataobject	boolean

De laatste kwantor is een aangepaste versie van de existentie kwantor, bij deze kwantor is het mogelijk om minimale en maximale hoeveelheden aan te geven. Deze hoeveelheden geven aan voor hoeveel individuele objecten de voorwaarde minimaal of maximaal moet gelden. In dit geval geeft m het maximum aan en n het minimum. Als een van beiden ontbreekt geldt alleen een maximum of minimum, als het maximum ontbreekt geldt dus een minimum zonder maximum en vice-versa.
Als voorbeeld voor deze functies kijken we naar een aantal van de regels die nodig zijn om de evaluatie van het concept samengesteld predicaat. Het samengesteld predicaat is het voornaamste concept waarin lijsten van onbepaalde lengte voorkomen en ook waar de kwantoren nodig zijn om de verschillende booleaanse waardes uit de verschillende subcondities te combineren.

Rule: SamengesteldPredicaat$_{true}^{alle}$		
Conclusie:		
SamengesteldPredicaat:	**Begintoestand:**	**Eindtoestand:**
- Quantificatie: alle	dataobject	True
- [Subconditie]:subconditie		
Voorwaarden:		
$\forall conditie: subconditie$	dataobject	True

De bovenstaande regel geeft een voorbeeld van het gebruik van een van de kwantoren, in dit geval de universele kwantor. Het gaat om een SamengesteldPredicaat, waarbij een lijst aan condities geevalueerd wordt en op basis van de hoeveelheid elementen uit de lijst die tot true evalueren besloten of het gehele SamengesteldPredicaat tot waar evalueert. Omdat de quantificatie in dit SamengesteldPredicaat ‘alle’ is, evalueert het gehele SamengesteldPredicaat enkel tot true als dit ook voor alle condities in de lijst zo is. Hierdoor wordt de universele kwantor op de lijst gebruikt, die evalueert in dit geval tot true als voor alle condities in de lijst subcondities ook geldt dat deze tot true evalueren gegeven het relevante dataobject.

3.3.6 Errors en Waarschuwingen:

Ook met Regelspraak kunnen fouten gemaakt worden, er kunnen regels geschreven worden die niet kloppen of geen correcte betekenis (kunnen) hebben. Hierdoor is het nodig om ook in Regelspraak te definiëren onder welke omstandigheden er error of waarschuwingen door het systeem gegeven worden.

In de normale specificatie voor natuurlijke semantiek zit dit niet expliciet verwerkt. Het is mogelijk om dit hier zonder bijzonderheden in te verwerken, hiervoor is het nodig om voor alle concepten te specificeren onder welke omstandigheden er een bepaalde error of waarschuwing gegeven wordt. Hierbij zouden de errors of waarschuwingen dan het resultaat van het uitvoeren van het concept zijn, en dit geeft twee problemen. Ten eerste zou dit het nodig maken dit resultaat ook te kunnen verwerken binnen andere concepten, wat ervoor zou zorgen dat voor elk concept een extra regel nodig is. Daarnaast leveren waarschuwingen naast de waarschuwing ook nog een daadwerkelijk resultaat op, waarnaast de waarschuwingen doorgegeven zouden moeten worden.

Om een onnodige wildgroei van regels om errors te verwerken te voorkomen verwerken we errors en waarschuwingen op hun eigen manier. Errors worden gespecificeerd in regels als de uitkomst van regels, waarschuwingen worden gespecificeerd in regels door in de tabel onder de uitkomst (het nieuwe dataobject of de evaluatie van conditie/expressie) de waarschuwing te specificeren. Het ‘doorgeven’ van errors of waarschuwingen naar onderliggende concepten wordt verder niet gespecificeerd, omdat dit nooit meer zal zijn dan simpelweg het doorgeven van de errors of waarschuwingen.

Als in het uitvoeren van een concept een error voorkomt dan wordt er uiteindelijk niet uitgevoerd en verandert er daarom ook niets aan het oorspronkelijke dataobject. De gebruiker moet in dat geval natuurlijk wel de error te zien krijgen. Als er in het uitvoeren een waarschuwing voorkomt, wordt de rest van de regel nog wel uitgevoerd en kan er dus een geldig veranderd dataobject ontstaan. Dit nieuwe dataobject is in dat geval de nieuwe toestand na het uitvoeren van het concept, ook als hier een waarschuwing in voorkomt, de waarschuwing wordt echter wel aan de gebruiker getoond.

Rule: SamengesteldPredicaat$_{tenminste}$		
Conclusie:		
SamengesteldPredicaat:	**Begintoestand:**	**Eindtoestand:**
– Quantificatie: ten minste n	dataobject	Error: “er zijn slechts “ + count (subconditie) + “ condities”
– [Subconditie]: subconditie		
Voorwaarden:		
n > count (subconditie)		

In de bovenstaande regel komt de mogelijkheid naar voren dat het concept SamengesteldPredicaat een error oplevert afhankelijk van de hoeveelheid condities in de lijst met subcondities. Om tot waar te evalueren zou voor dit SamengesteldPredicaat moeten gelden dat er ten minste n (waarbij n de hoeveelheid is) condities uit de lijst met subcondities tot waar moeten evalueren. In dit geval is de gegeven hoeveelheid n echter groter dan de hoeveelheid subcondities in de lijst met subcondities,

waardoor het gehele SamengesteldPredicaat nooit waar zou kunnen zijn, dit geeft een error. Hierdoor is de eindtoestand van het uitvoeren van deze regel dan ook de error met bijbehorende tekst.

4 VOORBEELDEN VAN CONCEPTEN:

Hierbij volgen een aantal voorbeelden van specificaties van verschillende concepten binnen Regelspraak om een beeld te geven van hoe volledige specificatie van deze concepten er uit zouden zien.

4.1 Voorbeeld ActieIndienVoorwaarde:

4.1.1 Syntax:

De syntax van ActieIndienVoorwaarde is ook bij de voorbeelden van Syntax voorgekomen. Binnen een ActieIndienVoorwaarde wordt eerst de Actie gegeven, vervolgens komt optioneel op een nieuwe regel de conditie als er voor de regel een conditie is.

```
ActieIndienVoorwaarde ::= Actie (¶ "indien" Conditie)?
```

Hieruit kunnen dus zowel instanties van ActieIndienVolgorde volgen die enkel een Actie hebben en instanties die zowel een actie als een concept hebben.

4.1.2 Concept vorm:

De conceptvorm bestaat dan ook uit het concept ActieIndienVoorwaarde met als subconcepten de Actie en de Conditie, zoals eerder ook als voorbeeld gebruikt.

ActieIndienVoorwaarde:

- actie: Actie
- conditie: Conditie

Hierbij zijn twee regels nodig om van de syntactische varianten van ActieIndienVoorwaarde conceptvormen te maken, een voor als er enkel een actie in het element staat, en een voor wanneer er ook een conditie in staat:

```
ActieIndienVoorwaarde: a1
```

↓

ActieIndienVoorwaarde:

- Actie: a1
- Conditie: *Geen*

```
ActieIndienVoorwaarde: a1 "¶ indien" c1
```

↓

ActieIndienVoorwaarde:

- Actie: a1
- Conditie: c1

4.1.3 Semantiek:

Om de semantiek van de ActieIndienVoorwaarde vast te leggen hebben we drie afleidingsregels nodig. De eerste is nodig voor de conceptvorm waarbij de conditie Geen is, in dit geval is er geen conditie gegeven en wordt de Actie dus sowieso uitgevoerd. Dan zijn twee regels nodig voor de gevallen waarbij er wel een conditie gegeven is: een regel voor als de conditie gegeven het dataobject tot true evalueert, in welk geval de actie uitgevoerd wordt en invloed kan hebben op het gegeven dataobject, en een regel voor als de conditie gegeven het dataobject tot false evalueert, in welk geval de actie niet uitgevoerd wordt en het dataobject dus onveranderd zal blijven. Dit levert in volgorde de volgende drie regels op:

Rule: ActieIndienVoorwaarde$_{geen}$		
Conclusie:		
SamengesteldPredicaat:	**Begintoestand:**	**Eindtoestand:**
- Actie: a1	dataobject	dataobject'
- Conditie: geen		
Voorwaarden:		
Actie: a1	dataobject	dataobject'

In deze eerste regel is er geen conditie gegeven, dus wordt de actie sowieso uitgevoerd, hierdoor is de enige voorwaarde dat als de actie tot een bepaalde wijziging in het dataobject leidt komt deze wijziging ook terug in de eindtoestand.

Rule: ActieIndienVoorwaarde$_{true}$		
Conclusie:		
SamengesteldPredicaat:	**Begintoestand:**	**Eindtoestand:**
- Actie: a1	dataobject	dataobject'
- Conditie: c1		
Voorwaarden:		
Actie: a1	dataobject	dataobject'
Conditie: c1	dataobject	true

In de tweede regel is wel een conditie gegeven, hier is de uitkomst hetzelfde, als het uitvoeren van de actie leidt tot een wijziging in het dataobject wordt deze ook in de eindtoestand van de hele ActieIndienVoorwaarde teruggevonden. Echter is hier een extra voorwaarde bij dat de conditie in deze toestand ook tot true evalueert. Is dit niet het geval dan is de derde regel van toepassing:

Rule: ActieIndienVoorwaarde$_{false}$		
Conclusie:		
SamengesteldPredicaat:	**Begintoestand:**	**Eindtoestand:**
- Actie: a1	dataobject	dataobject
- Conditie: c1		
Voorwaarden:		
Conditie: c1	dataobject	False

In deze regel is er wel een conditie gegeven, echter geldt deze regel als de conditie gegeven het dataobject tot false evalueert. Hierdoor zal de actie ook niet uitgevoerd worden en geen invloed op het dataobject kunnen hebben, hierdoor hoeft hier ook geen voorwaarde aan verbonden te worden. Omdat

de actie die het dataobject zou kunnen beïnvloeden niet uitgevoerd wordt is in de eindtoestand het dataobject hetzelfde als in de begintoestand.

4.2 Voorbeeld Gelijkstelling:

4.2.1 Syntax:

In de syntax van de gelijkstelling zien we terug dat er voor een concept soms meerdere tekstuele mogelijkheden zijn die convergeren tot hetzelfde concept in de conceptvorm, de syntax voor de gelijkstelling is:

```
Gelijkstelling ≔ Selectie (“moet gesteld worden op” | “moet berekend worden als”) Expressie
```

Hier zien we dat zowel de tekst “moet gesteld worden op” als “moet berekend worden als” hierin voor kunnen komen, echter leiden deze in de werkelijkheid tot dezelfde gelijkstelling.

4.2.2 Concept vorm:

De gelijkstelling heeft twee subconcepten, de selectie links en de expressie rechts, dit geeft de conceptvorm:

Gelijkstelling:

- links: Selectie
- rechts: Expressie

In de regels voor de conceptvorm hebben we een regel nodig voor beide taalvarianten, deze zullen echter tot hetzelfde concept leiden:

```
Gelijkstelling: links “moet gesteld worden op” rechts
```

↓

Gelijkstelling:

- Selectie: links
- Expressie: rechts

```
Gelijkstelling: links “moet berekend worden als” rechts
```

↓

Gelijkstelling:

- Selectie: links
- Expressie: rechts

4.2.3 Semantiek:

In dit geval is er maar een semantiek regel nodig (die eerder ook als voorbeeld gebruikt is), enerzijds omdat de uitvoering van gelijkstelling voor de invullingen van de subconcepten hetzelfde werkt en anderzijds doordat de conceptvorm de taalvariaties in de uitdrukkingen van het concept weggehaald heeft.

Rule: Gelijkstelling		
Conclusie:		
Gelijkstelling:	**Begintoestand:**	**Eindtoestand:**
- Selectie: links	dataobject	Dataobject[selectie ↦ waarde]
- Expressie: rechts		
Voorwaarden:		
Selectie: links	dataobject	Selectie
Expressie: rechts	dataobject	Waarde

In de gelijkstelling wordt het dataobject aangepast door de waarde van de expressie af te leiden gegeven het huidige dataobject en door deze waarde toe te wijzen aan het geselecteerde attribuut of kenmerk van het dataobject.

4.3 Voorbeeld PlusExpressie

4.3.1 Syntax

De PlusExpressie wordt gebruikt om de waardes van twee expressies bij elkaar op te tellen. In de syntax van de PlusExpressie komen hierdoor dan ook twee expressies voor als subconcepten, deze worden verder enkel gekoppeld door de ‘plus’ term:

```
PlusExpressie ≔ Expressie “plus” Expressie
```

4.3.2 Concept vorm

Door de simpele syntax van het concept is ook de vertaling naar de conceptvorm simpel te bevatten in een enkele regel. Hierbij wordt het belang van namen voor de verschillende subconcepten echter wel duidelijk, doordat beide subconcepten Expressies zijn maken de namen hiervoor het mogelijk deze expressies uit elkaar te houden:

```
PlusExpressie: links “plus” rechts
```

↓

PlusExpressie:

- Expressie: links
- Expressie: rechts

4.3.3 Semantiek

Voor de PlusExpressie is een enkele semantiekregel voldoende om de semantiek te geven:

Rule: PlusExpressie		
Conclusie:		
PlusExpressie:	**Begintoestand:**	**Eindtoestand:**
- Expressie: links	dataobject	waardel + waarder
- Expressie: rechts		
Voorwaarden:		
Expressie: links	dataobject	waardel
Expressie: rechts	dataobject	waarder

In de werking van het concept worden de waardes uit de verschillende subconcepten gecombineerd. Het resultaat van een plusexpressie is de som van de waardes waartot de subconcepten geevalueerd worden. De waardes van de subconcepten worden eerst individueel geevalueerd en worden daarna in de eindtoestand gecombineerd om de uiteindelijk waarde te geven waartoe de volledige PlusExpressie evalueert.

4.4 Voorbeeld KenmerkToekenning

4.4.1 Syntax

De kenmerktoekenning stelt voor objecten van een bepaald kenmerken vast. Hiermee is deze actie vergelijkbaar met de gelijkstelling in dat de hoofdzakelijke werking van het concept data van het dataobject in de invoer aanpassen is. In de syntax van de KenmerkToekenning is een taalvariatie terug te vinden afhankelijk van of het kenmerk waar het over gaat bezittelijk of persoonlijk is. De syntax voor het concept is dan:

```
KenmerkToekenning ≔ OnderwerpExpressie (“heeft” | “is een”) Kenmerk
```

4.4.2 Concept vorm

De vertaling naar conceptvorm zorgt wer ook bij de KenmerkToekenning weer voor dat de taalvariatie verdwijnt omdat deze geen impact heeft op de uiteindelijke semantiek van het concept. Hier zijn dus weer twee regels voor:

```
KenmerkToekenning: onderwerp “heeft” kenmerk
```

↓

KenmerkToekenning:

- OnderwerpExpressie: onderwerp
- Kenmerk: kenmerk

```
KenmerkToekenning: onderwerp “is een” kenmerk
```

↓

KenmerkToekenning:

- OnderwerpExpressie: onderwerp
- Kenmerk: kenmerk

4.4.3 Semantiek

De semantiek van de KenmerkToekenning is vrij simpel, de waarde van geselecteerde kenmerk van het dataobject in de input wordt op waar gezet. Dit wordt vaak gedaan afhankelijk van de evaluatie van een of meerdere condities, maar de evaluatie hiervan vind plaats in de ActieIndienVoorwaarde die dit zal omsluiten. Omdat het kenmerk enkel een selectie is die niet geevalueerd hoeft te worden heeft de semantiek van de Kenmerktoekenning geen voorwaarden en is het dus een axioma. Hiermee is de resulterende semantiek dus vrij simpel:

Rule: KenmerkToekenning		
Conclusie:		
KenmerkToekenning:	**Begintoestand:**	**Eindtoestand:**
- OnderwerpExpressie: onderwerp	dataobject	Dataobject[kenmerk ↦ true]
- Kenmerk: kenmerk		

5 OPSTELPROCES VAN SPECIFICATIE VAN CONCEPTEN

Om de specificatie van een concept te maken is het nodig om drie gerelateerd delen te specificeren: de grammaticale syntax, de regels die alle vormen van deze grammatica om kunnen zetten in de conceptvorm, en de semantiekregels die voor alle mogelijk invullingen van deze concepten de betekenis bepalen.

Hoe deze drie delen voor een taalelement opgesteld moeten worden is een algemeen vraagstuk, het moet zowel mogelijk zijn om dit te doen voor nieuwe elementen als voor taalelementen waar een implementatie voor bestaat. Voor beide soorten is het mogelijk dit algemeen te doen, maar voor elementen waarvoor een implementatie bestaat is het ook nuttig om te kijken naar een manier waarop dit op de implementatie gebaseerd kan worden om dit beter op elkaar aan te laten sluiten. Hier bespreken we een algemene methode om specificaties op te stellen, in de bijlage later bespreken we ook hoe dit gedaan kan worden aan de hand van de implementatie in ALEF.

Om in het algemeen een specificatie van een concept te maken kijken we eerst naar het opstellen van de conceptvorm van het element. Voor de conceptvorm is het namelijk alleen te kijken welke subconcepten in het element (moeten) zitten. Dit zijn de delen van het element die gerepresenteerd worden door een ander concept als deel van het concept op te nemen. Hierna is met de naam en de subconcepten met een eigen naam de conceptvorm al gemaakt.

Hierna kan naar de syntax gekeken worden. Hierbij gaat het om het vinden van alle vormen/schrijfwijzen die een concept hebben, waarbij alle taalkundige variaties via opties in de

grammatica opgenomen moeten worden. Hierbij gaat het ook om welke delen van de uitdrukkingen van het concept eigenlijk uit subconcepten bestaan, deze moeten op de plaatsen in de grammatica door de relevante non-terminals van deze subconcepten ingevuld worden.

Hierbij zijn dan regels nodig die voor alle syntaxvariaties die mogelijk zijn aangeven hoe deze variaties naar de conceptvorm omgezet kunnen worden. Deze regels zijn echter in de meeste gevallen enorm simpel, enkel het concept en de subconcepten blijven over, als het in de syntax mogelijk is om vormen te maken waarin een bepaald subconcept niet voor komt zal deze vanuit deze vorm in de conceptvorm ‘geen’ worden.

Hierna is het nodig om vanuit de conceptvorm de semantiekregels op te stellen die bepalen welk effect het gebruik van deze taalelementen in de taal heeft. Hierbij is het belangrijk eerst te bedenken welke verschillende regels nodig zijn, verschillende regels zijn nodig voor de combinaties waardes van subconcepten die tot andere uitkomsten leiden die niet simpelweg als parameterveranderingen weer te geven zijn. Dan moet er gekeken worden waar tot wat voor soort eindtoestand de regels moeten leiden, afhankelijk van het concept is dit een wijziging in het dataobject maar het kan bijvoorbeeld ook tot een boolean leiden in het geval van de evaluatie van een conditie.

Dan kunnen de regels daadwerkelijk opgesteld worden door te kijken hoe het evalueren van het concept met een bepaalde arbitraire input tot een bepaalde eindtoestand leidt en welke rol de subconcepten hierin spelen. Om vanuit waardes voor subconcepten regels op te stellen is het nuttig om voorwaarden op basis van subconcepten te maken door stelling te maken als: ALS het subconcept tot deze waarde evalueert DAN evalueert het hoofdconcept tot deze waarde. Hierbij kunnen ook aanvullen eisen zijn, zoals dat een evaluatiewaarde van een concept aan een voorwaarde voldoet, deze eisen moeten dan aan de voorwaarden toegevoegd worden. Op basis van de voorwaarden en eisen is ook de eindtoestand voor het hoofdconcept te bepalen.

Als een concept geen subconcepten heeft is het vaak veel simpeler om de regel(s) voor het concept op te stellen. De evaluatie kan direct van het concept naar een eindtoestand, eventueel met gebruik van het dataobject. Het kan hierbij alsnog noodzakelijk zijn meerdere regels te maken op basis van voorwaarden als resultaten uit het dataobject tot verschillende conclusies kunnen leiden. Door de syntax, conceptvorm, en semantiekregels op te stellen is de specificatie van een bepaald taalconcept compleet.

6 CONCLUSIE

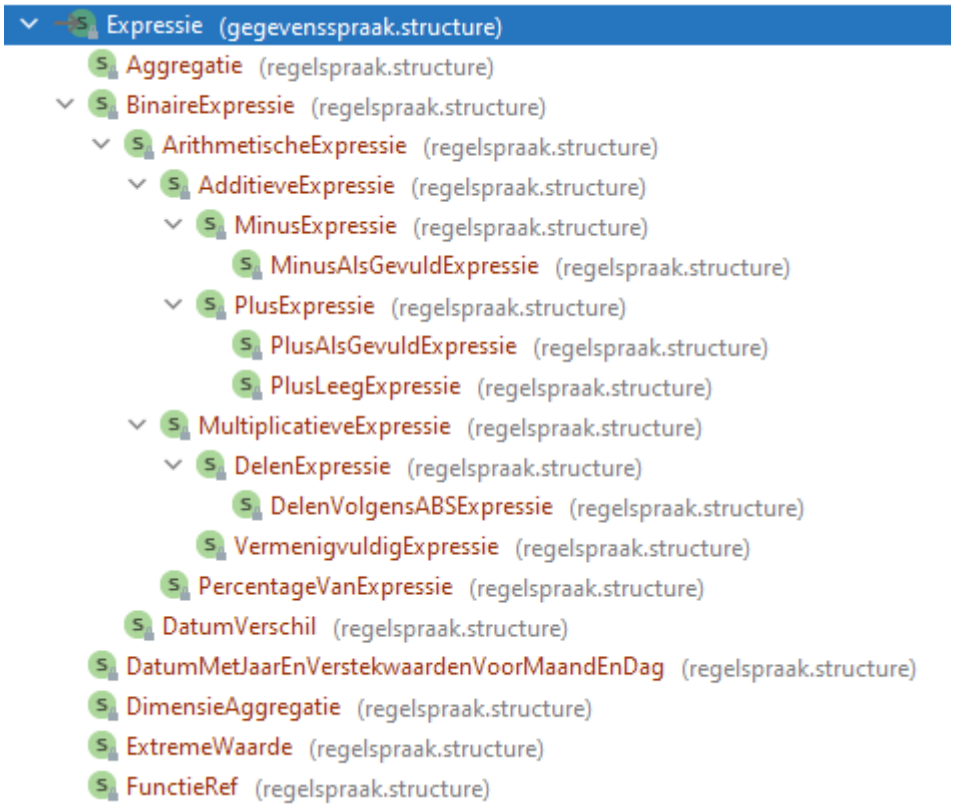
Met de methode die we in dit onderzoek voorstellen kan op effectieve manier een formele specificatie gemaakt worden van RegelSpraak waarin zowel speciceerd wordt welke vormen regels aan kunnen nemen als wat dit precies betekent. Door op deze manier een specificatie te maken van alle concepten binnen RegelSpraak kan voor de hele taal de syntax en semantiek vastgelegd worden op een manier die exact duidelijk maakt welk effect het uitvoeren van regels op data heeft.

Dit levert ook nog een aantal vragen op, zowel over deze specificatievorm als over het specificeren van RegelSpraak in het algemeen. Voor de specificatievorm is een belangrijke vraag om te kijken naar de begrijpelijkheid van de resulterende specificaties, zijn de specificaties goed te begrijpen door de mensen die ze moeten begrijpen ten behoeve van een goede implementatie? Omdat we dit niet geevalueerd hebben is het moeilijk in te schatten hoe goed deze specificaties daadwerkelijk te begrijpen zijn. Hiernaast is het belangrijk verder te kijken naar de manier waarop een specifcatie van de gehele taal gemaakt kan worden. Een huidig nadeel is dat het maken van een specificatie die exact aansluit bij het geïmplementeerde gedrag erg veel kennis van de werking van ALEF vereist om dit goed te vertegenwoordigen. Hierdoor kost het meer tijd om een volledige specificatie van RegelSpraak te maken, wat sowieso al een tijdrovend process is. Vandaar dat er ook verder gekeken kan worden naar tijdsefficiente manieren om zulke specificaties te maken.

In het opstellen van een daadwerkelijke specificatie moet ook duidelijk gekeken worden hoe een dergelijke specificatie onderhouden wordt. Specificatie en implementatie moeten dicht bij elkaar blijven, als dit te ver uit elkaar loopt kan er niet meer op de specificatie vertrouwd worden. Als een specificatie opgesteld wordt moet dus ook bekeken worden hoe de kwaliteit van de specificatie geborgen kan worden, ook op de wat langere termijn.

7 BIJLAGE 1: CONNECTIE MET ALEF:

In het maken van deze specificatievorm is veel van de informatie over welke vormen noodzakelijk, gepast en nuttig zijn gebaseerd op informatie uit ALEF. Zowel voor de verschillende taalelementen in Regelspraak als voor de betekenis en verschillende vormen van deze taalelementen is voornamelijk naar de implementatie in ALEF gekeken. Omdat de taalelementen centraal staan in het specificeren van Regelspraak leggen we uit hoe deze aan de implementatie in ALEF gerelateerd zijn.
De genoemde 'taalelementen' op basis waarvan de syntax gespecificeerd wordt zijn gelijk aan de concepten die in ALEF voorkomen. Voor elk concept in ALEF zijn er productieregels die aangeven welke taal het concept in een regel kan genereren. Deze productieregels worden gebaseerd op de structuur en de editor die voor het concept in ALEF gedefinieerd zijn.
In ALEF bestaan twee soorten concepten, reguliere concepten die teruggevonden worden in regels en abstracte concepten. Deze abstracte concepten bestaan om als algemene term boven een aantal reguliere concepten te staan, hiermee kan er in een keer gerefereerd worden naar de verschillende soorten reguliere concepten die op een bepaalde plaats voor kunnen komen.



Figuur 10: Abstract concept Expressie met een deel van de bijbehorende sub-concepten

Neem bijvoorbeeld het abstracte concept 'Expressie', er zijn geen instanties van het concept Expressie, maar er zijn wel veel andere concepten die een subconcept van Expressie zijn en als Expressie kunnen dienen als er een expressie als kind van een ander concept verwacht wordt. Een abstract concept

levert nooit instanties aan een regel, toch is het voor de syntaxspecificatie belangrijk om ook syntaxelementen voor de abstracte concepten in ALEF te maken. Hier kan naar gerefereerd worden wanneer een ander concept een kind van het abstracte concept heeft en de productieregels sturen de syntax effectief enkel door naar de subconcepten van het abstracte concept.

7.1 Specificatie uit ALEF:

Voor bestaande taalelementen die een implementatie hebben is het mogelijk en waarschijnlijk wenselijk om de specificatie te maken op basis van de implementatie in ALEF. Door dit te doen is het te voorkomen dat specificatie en implementatie uit elkaar lopen. Hierbij is het belangrijk om te weten waar de verschillende elementen die voor de specificatie van een concept nodig zijn in ALEF terug te vinden zijn.

Voor het vinden van syntax uit de implementatie in ALEF zijn de belangrijke plaatsen om te kijken de editor en indien aanwezig het linguistics aspect van het concept. Linguistics geeft de syntax over het algemeen iets duidelijker weer, echter is linguistics niet voor ieder concept beschikbaar. Hieruit valt af te lezen welke vormen uitdrukkingen van het concept kunnen hebben, en waar subconcepten hierin voorkomen.

In de structuurbestanden van de concepten zijn de conceptvormen grotendeels te vinden door de properties en children van de concepten als de subconcepten te zien. Al is het in gevallen dat concepten bovenliggende concepten uitbreiden of interfaces implementeren mogelijk dat er ook naar de structuur van deze concepten gekeken moet worden om alle subconcepten van het concept te vinden. De semantiek van concepten is voornamelijk te vinden in de interpreter/translator, voor de meeste concepten zal dit het bestand RegelspraakInterpreter te vinden zijn, echter zijn ook veel concepten die in regels terugkomen binnen Gegevensspraak geïmplementeerd en is de semantiek hiervan terug te vinden in de GegevensspraakInterpreter. De semantiek van een concept is te vinden in de functie van de node van het concept met een context. Hierbij komt wel dat hier ook een aantal ALEF-implementatie specifieke in deze functies voorkomen, zaken waar we bij de specificatie van regels over abstraheren. Uiteindelijk gaat het bij de semantiek om het beantwoorden van een aantal vragen op basis van de implementatie:

- Wat gebeurt er met het onderwerp/de data, door objecten aan te passen kan de data aangepast worden waardoor de eindtoestand ook aangepast wordt.
- Als de evaluatie van het concept tot iets anders dan (gewijzigde) data leidt, hoe wordt het eindresultaat dan bepaald.
- Worden de subconcepten geëvalueerd en waar worden de resultaten van deze voor gebruikt in het bepalen van de uitkomsten van de functie.
- Zijn er casusonderscheidingen in de functie die het nodig maken om verschillende semantiekregels voor het element te maken?

Door naar deze vragen te kijken is het mogelijk om voor het taalelement de nodige informatie te krijgen om de semantiekregels op te stellen. Waarmee deze informatieverzameling gecombineerd wordt met de algemene methode om specificaties van taalelementen te maken is het mogelijk om specificaties van concepten te maken die goed aansluiten op een bestaande implementatie, al vergt dit wel kennis van de werking van ALEF en hoe de code hiervan werkt.

OPEN UP
NEW HAN_UNIVERSITY OF APPLIED SCIENCES
HORIZONS.