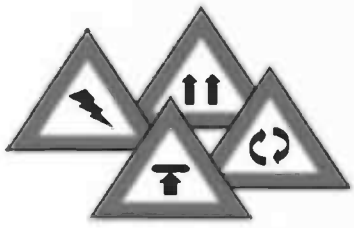
# VERIFICATIE VAN KENNISBANKEN

- Afstudeerscriptie -

M.F. van Santen de Hoog
studentnummer: 0957313
april 2005



Interne begeleider: dr. L.C. Verbrugge (Kunstmatige Intelligentie RuG)
Externe begeleiders: ing. M. Mastop msc. (Everest BV)
ir. O. de Groote (Everest BV)

**Kunstmatige Intelligentie**
**Rijksuniversiteit Groningen**

# SAMENVATTING

Het vakgebied van kennissystemen is voortgekomen uit de kunstmatige intelligentie, ongeveer in de jaren '60 en '70. Kennissystemen (ook wel expertsystemen genoemd) kenmerken zich door een opzet waarbij de kennis, de kennisbank (Knowledge Base, KB), gescheiden is van de afleidlogica. De kennis in de KB is meestal gerepresenteerd in if-then-regels of in beslistabellen, hoewel ook andere representatievormen mogelijk zijn.

Het bedrijf Everest BV te 's-Hertogenbosch is gespecialiseerd in het bouwen van kennissystemen. Everest BV is geïnteresseerd in technieken die het mogelijk maken de KB automatisch te verifiëren. Hiermee wordt bedoeld het detecteren van logische anomalieën zoals: redundantie (overbodige kennis), contradictie (tegenstrijdige kennis), circulariteit en onvolledige kennis. Anomalieën kunnen ertoe leiden dat het kennissysteem onjuiste of onvolledige output geeft. Verificatie is daarom een belangrijk onderdeel van het ontwikkelproces.

De hoofdvraagstelling van dit afstudeeronderzoek was of verificatietechnieken voor de systemen die Everest BV gebruikt mogelijk en haalbaar zijn. Hiertoe is theoretisch literatuuronderzoek gedaan naar verificatietechnieken, er zijn een aantal reeds bestaande verificatietools onderzocht, er zijn algoritmen opgesteld en uiteindelijk is een prototype verificatietool geconstrueerd.

In eerste instantie is het onderzoek toegespitst op de verificatie van zowel regelgebaseerde als beslistabelgebaseerde kennissystemen. Later bleek echter dat het eenvoudiger was om de algoritmiek en het prototype alleen op regels te baseren. Beslistabellen kunnen worden geverifieerd door deze eerst om te zetten in regels.

Het uiteindelijk geconstrueerde prototype is in staat om alle categorieën en subcategorieën van anomalieën te detecteren en geeft hiervan een uitgebreide rapportage. Hoewel het prototype niet gebouwd is voor alle complexiteiten en details van de systemen die Everest BV bouwt, is de doelstelling op dit punt geslaagd. Naast de mogelijkheid van verificatie ging het echter ook om haalbaarheid. Een belangrijk punt hierbij is de snelheid van anomaliedetectie in praktijksituaties. Voor de meeste anomaliecategorieën verliep deze binnen aanvaardbare normen. Echter, bij bepaalde specifieke categorieën was sprake van een exponentiële toename van de detectiesnelheid, naarmate de complexiteit van de input toenam. In praktijksituaties was de snelheid van deze specifieke tests niet meer aanvaardbaar. Enig onderzoek en denkwerk resulteerden echter in een aantal concrete oplossingen voor dit probleem waardoor het prototype in de toekomst ook bij deze situaties inzetbaar kan zijn.

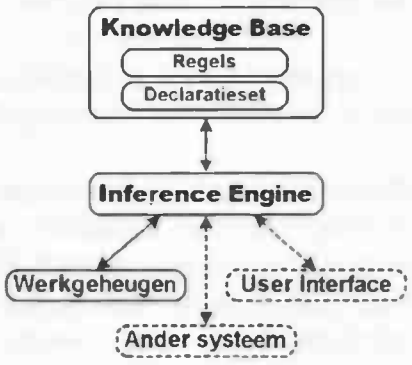
# INHOUDSOPGAVE

# H1 INLEIDING

## 1.1 Kennissystemen

Het vakgebied van kennissystemen is voortgekomen uit de kunstmatige intelligentie in zijn beginperiode, zo ongeveer in de jaren '60 en '70 [1]. Kennissystemen werden destijds vaak expertsystemen genoemd, aangezien getracht werd de kennis en expertise van een menselijke expert in een automatisch computersysteem te vatten. De ontwikkeling, bouw en werking van kennissystemen was radicaal anders dan die van de destijds gebruikelijke computersystemen. In plaats dat voor elke mogelijke situatie functies of procedures worden geschreven, worden in kennissystemen alleen de onderliggende regels beschreven. Daarnaast is er een aparte afleidmodule, de inference engine, die op basis van de regels tot een conclusie of een actie komt.

De algemene opzet van een kennissysteem is te zien in Figuur 1. De basis van ieder kennissysteem wordt gevormd door de kennisbank, ook vaak genoemd: *Knowledge Base* (KB). In de KB bevinden zich de regels die bepalend zijn voor het gedrag van het systeem en de uitkomsten die het systeem biedt. Er bestaan verschillende representatievormen van deze regels. Voorbeelden hiervan zijn productieregels (if-then-regels), beslistabellen, beslisbomen en cases. Daarnaast bevindt zich in de KB een declaratieset, wat eigenlijk niets anders is dan een set van feiten. Hierin wordt aangegeven wat de begincondities zijn, welke beperkingen het systeem kent (constraints), en wat de doelen zijn. De inference engine maakt gebruik van het werkgeheugen om tussenresultaten op te slaan. Daarnaast is er eventueel communicatie met de gebruiker via een user interface, en er kan eventueel communicatie zijn met een ander (omvattend) systeem. Ten opzichte van conventionele softwaresystemen kennen kennissystemen een aantal belangrijke voordelen. Doordat de regels en declaraties in de KB gescheiden zijn van de rest van het systeem, kunnen deze gemakkelijk gewijzigd, verwijderd of toegevoegd worden, zonder dat de rest van het systeem daarvoor aangepast hoeft te worden. Bovendien maakt de volgorde van de regels niet uit (de Church Rosser eigenschap [9]). Kennissystemen zijn hierdoor flexibeler.




**Figuur 1: Algemene opzet van een kennissysteem**

Kennissystemen werden in het begin vooral gebruikt in academische toepassingen (het medische diagnosesysteem MYCIN is een beroemd voorbeeld, ontwikkeld in de jaren '70). De voordelen van kennissystemen gingen echter niet geheel onopgemerkt voorbij aan het bedrijfsleven. Met name het argument dat kennissystemen over een grotere mate van flexibiliteit beschikken dan conventionele systemen, betekende dat er bespaard kon worden op de onderhoudskosten en wellicht ook op het ontwikkeltraject [1]. De toepassing van kennissystemen in het bedrijfsleven zorgde voor een uitbreiding van het domein. Voortaan was het tevens mogelijk om de kennis en expertise van meerdere personen en de overkoepelende processen te implementeren in één kennissysteem. In plaats van *regels* werd vaak gesproken over *business rules*.

## 1.2 Validatie en verificatie van de Knowledge Base

Bij een kennissysteem is de kwaliteit van de Knowledge Base (KB) is van groot belang. Ten eerste is het belangrijk dat de KB de benodigde kennis bevat om alle problemen op te lossen die aan het systeem gevraagd kunnen worden. Het systeem moet dus op alle mogelijke vragen of input met correcte antwoorden of acties komen. Ten tweede is het belangrijk dat de regels in de KB geen fouten of overbodigheden bevatten en een logisch consistent geheel vormen. Het toetsen van deze twee zaken wordt respectievelijk *validatie* en *verificatie* genoemd [1, 13].
Deze twee zaken worden in de literatuur nog al eens door elkaar gehaald en vaak worden ze in één adem genoemd als validatie & verificatie. Het is echter goed om een duidelijk onderscheid te maken. Bij validatie gaat het om de acceptatie van de toekomstige gebruiker. Voldoen het gedrag en de uitkomsten van het systeem aan zijn verwachtingen? Bij verificatie gaat het om de technische correctheid en efficiëntie van het systeem. De volgende twee uitspraken geven het onderscheid aan.

validatie: *"Building the right system"*
verificatie: *"Building the system right"*

Ondanks dat de techniek van kennissystemen als alternatief voor conventionele softwaresystemen inmiddels wordt toegepast binnen het bedrijfsleven, gaat deze opmars nog maar langzaam. Eén van de redenen hiervoor zou kunnen zijn dat veel bedrijven kennissystemen nog niet betrouwbaar voldoende achten [1, 6]. In de afgelopen decennia waarin de technieken van kennissystemen werden geoptimaliseerd, hebben bedrijven die conventionele software produceren eveneens bepaald niet stil gestaan. Inmiddels is de betrouwbaarheid en kwaliteit van conventionele software van zeer hoog niveau. De vele concurrentie die aanwezig is en tevens enkele juridische aspecten hebben hieraan bijgedragen. Het ontwikkeltraject is tegenwoordig zeer gestructureerd en kent vele testfasen en kwaliteitstests [1].

Om de betrouwbaarheid en kwaliteit van kennissystemen op een even hoog niveau te brengen als conventionele software dat inmiddels heeft, is een degelijke validatie en verificatie van de KB van groot belang. Zo vroeg mogelijk tijdens de ontwerpfase moeten validatie en verificatie worden toegepast. Het herstellen van fouten kost later vaak aanzienlijk meer tijd en geld. Maar validatie en verificatie is tevens van groot belang tijdens het gebruik van het systeem, wanneer er regels worden aangepast of toegevoegd. Wanneer het beheer en onderhoud in handen is van het bedrijf zelf, de klant dus, is dit extra belangrijk. Mensen uit het bedrijfsleven hebben immers geen achtergrond in de logica of in de kennistechnologie. Fouten en inconsistenties zullen onvermijdelijk aan de orde komen. Volgens de Business Rules Approach [5, 20] is het echter belangrijk dat bedrijven hun bedrijfsregels in eigen beheer hebben en zelf kunnen aanpassen wanneer dit nodig is. Regels kunnen snel worden aangepast zonder tussenkomst van derden, miscommunicatie kan vermeden worden en bovendien krijgt het bedrijf meer inzicht en overzicht in de eigen business rules. Het is duidelijk dat validatie en verificatie vanuit dit oogpunt een onmiskenbaar onderdeel in het proces zal zijn.

Een goede validatie van de KB kan gerealiseerd worden door gedurende het gehele ontwerpproces telkens de regels in de KB te vergelijken met die van menselijke

deskundigen, andere betrouwbare bronnen en de eisen van de opdrachtgever [1]. Een goede verificatie van de KB dient te geschieden door te testen op de aanwezigheid van zogenaamde anomalieën. Anomalieën kunnen wijzen op fouten of inefficiënties in de KB. Een korte beschrijving van mogelijke anomalieën volgt in de volgende paragraaf.

## 1.3 Mogelijke anomalieën in een Knowledge Base

Anomalieën in de KB kunnen worden geclassificeerd in een aantal categorieën [2]. Hieronder volgt een korte beschrijving van deze categorieën. Een uitgebreidere en meer formele beschrijving volgt in hoofdstuk 2.

1. **Redundantie:**
   Er mogen geen regels zijn die nooit enige contributie kunnen bijdragen aan de conclusies van het systeem. Er bestaan verschillende vormen:
   - Er mogen geen regels zijn die samengevat kunnen worden tot een reeds aanwezige regel en dezelfde regels mogen niet op meerdere plaatsen aanwezig zijn in de KB (subsumed rules).
   - Er mogen geen regels zijn die nooit gebruikt kunnen worden (onvuurbare regels).
   - Er mogen geen regels zijn waarvan de conclusie nooit gebruikt kan worden of waarvan de conclusie niet behoort tot de set van mogelijke doelen (onbruikbare conclusies).
2. **Inconsistentheid:**
   Er mogen geen contradicties zijn in de regels. De volgende twee regels zijn niet consistent: `if A then B` en `if A then ¬B` (en A is waar).
3. **Circulariteit:**
   Voor alle mogelijke input moet gelden dat bij het afleiden van de conclusie nooit mag gelden dat een regel vaker dan één keer wordt gebruikt. De volgende twee regels bevatten circulariteit: `if A then B` en `if B then A`.
4. **Onvolledigheid:**
   Voor alle mogelijke input moet gelden dat het systeem een conclusie kan geven. De volgende twee regels zijn niet volledig:
   `if A < 2 then B` en `if A > 3 then C` (en het domein van A bevat alle gehele getallen).

Wanneer een anomalie gedetecteerd wordt in de KB, hoeft dit niet altijd te betekenen dat deze gerepareerd moet worden. In sommige organisaties wordt er bijvoorbeeld bewust voor gekozen om een bepaalde overlap van regels in stand te houden en zo dus tevens redundantie in stand te houden. Ook kan het voorkomen, bijvoorbeeld in de medische wereld, dat één diagnose kan leiden tot meerdere, inconsistente mogelijke oplossingen. In deze gevallen beslist uiteindelijk de menselijke expert [14]. Tevens is het mogelijk dat een bepaalde anomalie bewust in stand gehouden wordt wegens in de nabije toekomst geplande uitbreidingen of wijzigingen die de anomalie zullen elimineren.

## 1.4 Het Knowledge Framework van Everest

Het afstudeeronderzoek naar aanleiding waarvan deze scriptie geschreven is, vond plaats bij het Nederlandse bedrijf Everest BV (hierna te noemen: Everest). Everest is een bedrijf dat zich heeft gespecialiseerd in het ontwikkelen van hoogwaardige kennissystemen voor haar klanten. Everest gebruikt hiervoor een door haarzelf ontwikkelde omgeving: het *Knowledge Framework*. De KB van het Knowledge Framework wordt onderverdeeld in business logica en fuzzy logica. De business logica kan vervolgens weer in de vorm zijn van beslistabellen en/of in de vorm van regels. Tijdens dit afstudeerproject werd er bovendien gewerkt aan het implementeren van beslisbomen. De inference engine beschouwt al deze bronnen in het vormen van een oordeel of het uitvoeren van een actie. In hoofdstuk 4 zal dieper worden ingegaan op de eigenschappen van het Knowledge Framework.

## 1.5 Probleemstelling

Er is inmiddels een redelijk aantal tools beschikbaar die de KB van een kennissysteem automatisch kunnen verifiëren op de anomalieën die zojuist genoemd zijn. Echter, vele van deze tools testen slechts op een gedeelte van de mogelijke anomalieën en niet op allemaal [16]. Bovendien is het gros van deze tools ontwikkeld in de academische wereld, waardoor deze praktisch en commercieel gezien niet bruikbaar zijn [11]. Everest is er in geïnteresseerd om in de toekomst automatische verificatietechnieken toe te passen in de kennissystemen die zij haar klanten levert en deze techniek dus commercieel te maken. Het toepassen van automatische verificatie kan zorgen voor kostenbesparing en bovendien een kwalitatief beter product. De hoofdvraag van dit afstudeerproject is daarom:

*Is het mogelijk om een prototype van een automatische verificatietool te construeren voor Everest dat in staat is een willekeurige KB te verifiëren op inconsistentie, onvolledigheid, redundantie en circulariteit*?

Om de hoofdvraag te kunnen beantwoorden zal eerst een verdergaand literatuuronderzoek gedaan moeten worden op dit gebied en zullen reeds bestaande tools bestudeerd moeten worden. Op basis van de kennis die hiermee wordt opgedaan kan een algoritme worden opgesteld en vervolgens een prototype geconstrueerd worden. Het prototype zal alleen anomalieën moeten kunnen detecteren en niet repareren. Men zou kunnen denken dat wanneer een anomalie automatisch gedetecteerd kan worden, dat deze dan tevens automatisch gerepareerd kan worden. In de praktijk is dit echter zeer moeilijk. Vaak is kennis van de wereld of van een menselijke expert nodig om een goede reparatie te kunnen verrichten [14].
Verder zal dit onderzoek en het te construeren prototype toegespitst zijn op alleen regels en beslistabellen. Hoewel anomalieën ook kunnen voorkomen andere kennisrepresentaties, zullen die omwille van de beperkte tijd en complexiteit van het probleem buiten beschouwing worden gelaten.

# H2 THEORETISCH KADER

In dit hoofdstuk wordt een gedegen theoretische basis gelegd voor de rest van dit afstudeeronderzoek. De theorie van kennisrepresentatie via regels en via beslistabellen zal eerst worden behandeld. Daarna volgt een uitgebreide analyse van alle verschillende typen van anomalieën die mogelijk zijn in kennissystemen en als laatste een globale beschrijving van de theorie van detectiemethoden.

## 2.1 Vormen van kennisrepresentatie

De regels in de KB van een kennissysteem kunnen op verschillende manieren worden gerepresenteerd. De meest voorkomende representatievormen zijn: productieregels, beslistabellen (decision tables of DT's) en beslisbomen.

Productieregels zijn niets anders dan if-then-regels. Als aan de condities van de regel wordt voldaan, dan kunnen de conclusies worden afgeleid (Figuur 2). In principe kunnen regels in natuurlijke taal worden gespecificeerd. Echter, om dubbelzinnigheid te voorkomen en om gebruik te kunnen maken van computers voor automatische afleidingen, worden ze meestal in een formele taal geschreven, bijvoorbeeld de propositielogica of de eerste orde predikatenlogica. Productieregels kunnen gebruikt worden voor zeer specifieke situaties door gebruik te maken van een logische combinatie van diverse verschillende condities. De expressiviteit van productieregels in de eerste orde predikatenlogica is zeer groot. Hier staat echter tegenover dat het berekenen van bepaalde zaken, zoals inconsistentie, een mogelijk onbeslisbaar probleem vormt, wanneer de volledige expressiviteit van de taal wordt gebruikt [9]. Vaak wordt om deze reden een subset van de eerste orde predikatenlogica gebruikt, die wel beslisbaar is.

Beslistabellen bieden een overzichtelijke vorm van het beslisproces (Figuur 2). Met name wanneer de kennis niet van zeer complexe aard is, kan één beslistabel dezelfde kennis bevatten als vele productieregels. Het is bewezen door Colomb en Chung dat beslistabelgebaseerde systemen en propositionele regelgebaseerde systemen formeel equivalent zijn [5]. In beslistabellen worden de condities gescheiden van de acties in aparte gedeelten van de tabel. De acties van tabellen hebben logisch dezelfde rol als de conclusies van regels. Beslistabellen bieden naast de overzichtelijke representatievorm het voordeel dat op gemakkelijke wijze gebruikgemaakt kan worden van meerwaardige logica. Naast de logische waarden 'waar' en 'onwaar' kan de waarde 'onbekend' worden gebruikt. In de meeste regelgebaseerde systemen is dit niet het geval.

Beslisbomen bieden een handige grafische representatie van het beslisproces. Deze representatie is bijvoorbeeld zeer geschikt in classificatieproblemen. Voor geautomatiseerde kennissystemen is deze representatievorm meestal minder geschikt. Beslisbomen kunnen in dat geval beter worden omgezet in beslistabellen.

**Regel:**

ALS (Speakertype = Single17) EN (Kameroppervlakte in $m^2$ < 30)
DAN (Gewenst versterkervermogen = >50 watt)

**Beslistabel:**

| Speakertype | Single17 | | Double17 | |
|---|---|---|---|---|
| Kameroppervlakte in $m^2$ | < 30 | >= 30 | < 30 | >= 30 |
| Gewenst versterkervermogen | >50 watt | >80 watt | >70 watt | >100 watt |

**Figuur 2: Voorbeeld van een regel en een tabel**

In sommige systemen wordt ook gebruikt gemaakt van fuzzy logica toegepast op casussen. Casussen zijn feitelijk voorbeelden van veelvoorkomende situaties. Casusgebaseerd redeneren houdt in dat een nieuwe situatie wordt vergeleken met bekende casussen. Er hoeft geen exacte match te zijn met een casus; fuzzy logica zorgt voor de selectie van de beste alternatieven.

In de volgende paragrafen zal een uitgebreidere beschrijving volgen van regelgebaseerde kennissystemen en van systemen op basis van beslistabellen. Er zal in deze scriptie niet verder ingegaan worden op beslisbomen en casussen. Deze systemen vallen buiten dit onderzoek. De reden hiervoor ligt in de beperkte tijd die voor dit onderzoek beschikbaar is en het feit dat door Everest is aangegeven dat anomaliedetectie in regels en beslistabellen de hoogste prioriteit heeft.

### 2.1.1 Regelgebaseerde kennissystemen

Hieronder volgen een aantal formele definities van een KB op basis van productieregels, naar aanleiding van Preece en Shinghal [2] en uitgaande van een subset van de eerste orde predikatenlogica.

Een KB bestaat uit een verzameling regels (Rule Base) en een declaratieset (Declaration Set):

1. **KB** = Rule Base $\cup$ Declaration Set

De Rule Base bestaat uit een set van regels geschreven in conjunctievorm*. De linkerzijde van een regel bestaat uit een conjunctie van condities (antecedenten), de rechterzijde uit een conjunctie van conclusies. Zowel de condities als de conclusies zijn literals:

2. **Rule Base (R)** = $\{R_1, R_2, \ldots, R_n\}$ met $R_i = L_{i1} \wedge \ldots \wedge L_{in} \rightarrow M_{i1} \wedge \ldots \wedge M_{im}$
   $\{L_{i1}, \ldots, L_{in}\}$ = condit($R_i$) en $\{M_{i1}, \ldots, M_{im}\}$ = concl($R_i$)

* *In principe zijn ook regels mogelijk waarin disjuncties voorkomen in de linker- of rechterzijde. De conjunctievorm kent echter een aantal voordelen wat betreft de afleidmethoden en is bovendien leesbaarder. Preece en Shingal hanteren in [2] om deze redenen zelfs de implicatieve Horn Clause vorm. Hierbij wordt de regel in conjunctievorm geschreven waarbij bovendien slechts één conclusieliteral is toegestaan. Regels waarin disjuncties voorkomen kunnen trouwens altijd worden omgeschreven in regels in de conjunctievorm of zelfs in de implicatieve Horn Clause vorm [17].*

De Declaration Set bestaat uit een set van goal literals, een set van input literals en een set van semantische contradicties (constraints).

3. **Declaration Set (D)** = $\mathbf{G} \cup \mathbf{I} \cup \mathbf{C}$
   $\mathbf{G} = \{G_1, \ldots, G_p\}$ de set goal literals (alle mogelijke output of conclusies)
   $\mathbf{I} = \{I_1, \ldots, I_q\}$ de set input literals (alle mogelijke input)
   $\mathbf{C} = \{C_1, \ldots, C_r\}$ de set semantische contradicties: $C_i = \{L_{i1}, \ldots, L_{in}\}$
   met $L_{i1} \wedge \ldots \wedge L_{in}$ is een semantische contradictie

Een semantische contradictie is een contradictie die niet rechtstreeks blijkt uit de syntaxis van de formule. Om dit soort contradicties te kunnen vaststellen is kennis van de betekenis (de semantiek) noodzakelijk. De formule $A \wedge B$ waarbij A betekent *"Het regent"* en B betekent *"Het is droog"*, is bijvoorbeeld geen syntactische contradictie maar wel een semantische contradictie. Semantische contradicties moeten dus van te voren expliciet worden gedefinieerd. Syntactische contradicties zijn direct af te leiden uit de vorm van de formule, zonder dat de betekenis van de literals bekend hoeft te zijn, bijvoorbeeld $A \wedge \neg A$.
Een voorbeeld van een Rule Base en declaratieset van een (zeer eenvoudige) KB is te zien in Figuur 3. In dit voorbeeld leidt regel 5 trouwens tot een semantische contradictie.

| Rule Base (R) |
|---|
| $R_1$: $A \rightarrow C$ |
| $R_2$: $B \rightarrow D$ |
| $R_3$: $C \wedge D \rightarrow E$ |
| $R_4$: $E \rightarrow F$ |
| $R_5$: $F \rightarrow G$ |

| Declaration Set (D) |
|---|
| **G** = {E, F} |
| **I** = {A, B} |
| **C** = {$C_1$ = {F, G}} |

**Figuur 3: Een eenvoudige KB**

Een omgeving (Environment) is een subset van alle input literals I die geen semantische contradictie vormen:

4. **Environment Set (E)**: $E \in \mathbf{E}$ iff $((E \subseteq \mathbf{I}) \wedge (\forall C \in \mathbf{C})(E \nvdash C))$
   waarbij $(A \nvdash B)$ betekent: B is niet logisch afleidbaar uit A.

De set van alle hypotheses is de set van alle literals in de conclusies van alle regels:

5. **Hypotheseset (H)**: $H \in \mathbf{H}$ iff $(\exists R \in \mathbf{R})(\text{concl}(R) = H)$

Het predikaat infer(H, **R**, E) betekent dat hypothese H afleidbaar is uit Rule Base **R** wanneer environment E als input wordt gegeven:

6. **Afleidbaar**: infer(H, **R**, E) iff $(\mathbf{R} \cup E) \vdash H$
   met: $H \neq \top$ (H is geen tautologie*).
   en waarbij $(A \vdash B)$ betekent: B is logisch afleidbaar uit A.

* *De beperking dat H geen tautologie mag zijn was niet aanwezig in de definitie van het predikaat infer van Preece en Shinghal [2], maar is strikt genomen wel noodzakelijk.*

De set van afleidbare hypotheses is de set van hypotheses die afgeleid kunnen worden vanuit een mogelijke environment E:

7. **Afleidbare hypotheses (A)**: $H \in \mathbf{A}$ iff $(\exists E \in \mathbf{E})$ (infer(H, **R**, E))

Het predikaat firable(R, **R**, E) betekent dat er een mogelijke environment E bestaat zodat de condities van regel R een logisch gevolg zijn van het toepassen van E als input op **R**:

8. **Vuurbare regel**: firable(R, **R**, E) iff $(\exists E \in \mathbf{E})( (\mathbf{R} \cup E) \vdash \text{condit}(R))$

### 2.1.2 Beslistabelgebaseerde kennissystemen

## Algemene structuur

Beslistabellen behoren tot de klasse van formele talen. In beslistabellen worden de condities (situaties in het domein) gescheiden van de acties (transformaties op het domein). Condities en acties zijn beide een eindige set. Het totale beslisproces wordt gevormd door één of meerdere beslistabellen, eventueel hiërarchisch gerelateerd.
Tabel 1 laat de algemene structuur van een beslistabel zien zoals beschreven in [8].

| DT | | Regels | | | | Else-regel | Complexiteit van testen/uitvoeren |
|---|---|---|---|---|---|---|---|
| | | $P_1$ | $P_2$ | ... | $P_n$ | E | |
| Condities | $C_1$ | $c_{11}$ | $c_{12}$ | ... | $c_{1n}$ | [ ] | $t_1$ |
| | $C_2$ | $c_{21}$ | $c_{22}$ | ... | $c_{2n}$ | [ ] | $t_2$ |
| | ... | ... | ... | ... | ... | [ ] | ... |
| | $C_m$ | $c_{m1}$ | $c_{m2}$ | ... | $c_{mn}$ | [ ] | $t_m$ |
| Acties | $A_1$ | $a_{11}$ | $a_{12}$ | ... | $a_{1n}$ | $a_{1, n+1}$ | $q_1$ |
| | $A_2$ | $a_{21}$ | $a_{22}$ | ... | $a_{2n}$ | $a_{2, n+1}$ | $q_2$ |
| | ... | ... | ... | ... | ... | ... | ... |
| | $A_k$ | $a_{k1}$ | $a_{k2}$ | ... | $a_{kn}$ | $a_{k, n+1}$ | $q_k$ |
| Gebruikfrequentie | | $f_1$ | $f_2$ | ... | $f_n$ | $f_E$ | |

**Tabel 1: Algemene structuur beslistabel**

Een kolom kan in feite gezien worden als een productieregel. Zo kan bijvoorbeeld de tweede kolom in Tabel 1 als volgt gelezen worden: Als $C_1 = c_{12}$ en $C_2 = c_{22}$ en ... en $C_m = c_{m2}$ doe dan $a_{12}$ vervolgens $a_{22}$ ... $a_{k2}$ ($A_1$ heeft de hoogste prioriteit). In beslistabellen wordt echter meestal niet gesproken over de 'conclusies' van een kolom, maar over de 'acties'. Een bepaalde conditie $c_{ij}$ kan in beslistabellen ook de waarde * hebben. Dit betekent dat de waarde niet van belang is en onbekend mag zijn (meestal wordt een * een *don't care* genoemd). Beslistabellen ondersteunen dus een meerwaardige logica, in tegenstelling tot de meeste regelgebaseerde systemen. Over het algemeen wordt getracht de beslistabel zodanig te construeren dat de condities die minder afhankelijk zijn van andere condities, hoger in de tabel staan. Zo wordt de leesbaarheid van de tabel vergroot.

In Tabel 1 wordt tevens een speciale ELSE-kolom aangegeven. Meestal wordt hiervoor in de conditierij het symbool [] gebruikt. Een ELSE-kolom is niet verplicht. Ook bestaat de mogelijkheid om alleen in bepaalde condities $c_{ij}$ een else conditie [] op

te nemen. In het voorbeeld van Tabel 1 wordt bovendien nog aangegeven bij elke conditierij wat de complexiteit is van het testen van de condities en het uitvoeren van de acties. Dit kan gebruikt worden door het zoekalgoritme om de snelheid van het zoeken te verhogen of voor een eventuele kostenberekening. Daarnaast wordt in Tabel 1 tevens de gebruiksfrequentie aangegeven bij elke regel. Deze kan de betrouwbaarheid van een regel aangeven. De complexiteit en de gebruiksfrequentie zijn in grijs aangegeven omdat deze slechts optioneel zijn en in de meeste beslistabelsystemen niet aan de orde zijn.

## Meervoudige tabelsystemen

Meervoudige tabelsystemen kunnen bestaan uit losse tabellen die niets met elkaar te maken hebben maar ook subtabellen die de condities van een hoofdtabel nader specificeren [6, 7]. Tabellen 2 en 3 laten een voorbeeld zien van een hoofdtabel waarvan één van de condities afhankelijk is van een andere tabel, die ook wel de conditionele subtabel wordt genoemd. Er is een conventie om een attribuut dat afgeleid wordt in een andere tabel weer te geven met ^Attribuutnaam.

| Salaris | < 40.000 | | >= 40.000 | |
|---|---|---|---|---|
| Erfenis | ja | nee | ja | nee |
| Budget | klein | klein | groot | klein |

**Tabel 2: Conditionele subtabel**

| ^Budget | klein | | groot | |
|---|---|---|---|---|
| Goede klant | ja | nee | ja | nee |
| Uitvoeren order | ja | nee | ja | ja |

**Tabel 3: Hoofdtabel**

Elke conditionele subtabel kan op zijn beurt weer een hoofdtabel zijn van een andere conditionele subtabel, zodat uiteindelijk een meer complex systeem van subtabellen gevormd kan worden. Vantienen en anderen spreken bovendien over actie subtabellen [6, 7] die een nadere specificatie geven van een actie van een hoofdtabel. Feitelijk is dat echter enigszins overbodig: een actietabel kan gewoon gezien worden een nieuwe hoofdtabel en de tabel waarvan één van de acties nader wordt gespecificeerd heeft hiervoor de functie van conditionele subtabel.

## Definities

Wanneer het werk van Preece, Eremeev en Vanthienen [2, 6, 7, 8] wordt gecombineerd, kunnen de volgende formele definities met betrekking tot beslistabelgebaseerde systemen worden vastgesteld.

Een KB bestaat uit een verzameling beslistabellen (DT Base) en een declaratieset (Declaration Set):

1. **KB** = DT Base $\cup$ Declaration Set

De DT Base bestaat uit een set van beslistabellen:

2. **DT Base (DT)** = $\{DT_1, \ldots, DT_n\}$

Een beslistabel (DT) bestaat uit de volgende onderdelen:

3. **Decision Table** (DT):

- Een set van conditie onderwerpen **CS** = $\{CS_1, \ldots, CS_n\}$.
- Een set van conditie domeinen **CD** = $\{CD_1, \ldots, CD_n\}$.
  $CD_i$ geeft aan wat alle mogelijke waarden zijn van $CS_i$.

- Een set van conditiekolommen **CK** = $\{CK_1, \ldots, CK_k\}$
  met $CK_i = \{C_{i1}, \ldots, C_{in}\}$ is een conditiekolom waarbij $C_{ij}$ = een conditiewaarde (uitgedrukt in een logische expressie).
- Een set van actieonderwerpen **AS** = $\{AS_1, \ldots, AS_m\}$
- Een set van actiekolommen **AK** = $\{AK_1, \ldots, AK_k\}$
  met $AK_i = \{A_{i1}, \ldots, A_{im}\}$ is een actiekolom waarbij $A_{ij}$ = een actiewaarde

De definities voor Declaration Set, Environment, Hypothese en Afleidbare hypothese zijn gelijk aan de definities voor regelgebaseerde systemen. Deze zullen hier niet worden herhaald. Ze zijn voor de volledigheid opgenomen in Appendix A.

4. **Toepasbare situaties:** De set van conditievectoren die behoren tot de conditietabel van een DT.

Een herkende datavector (recognized datavector) is een datavector die matcht met een kolom in de conditietabel.

5. **Recognized datavector**[*]**:** Datavector $S = \{S_1, \ldots, S_m\}$ wordt herkend door de DT *T* als $(\exists CK_i \in \mathbf{CK})(\forall C_{ij} \in CK_i)(C_{ij} \neq * \rightarrow C_{ij} \in S)$

   In dit geval geldt dan: $S \rightarrow P_i$ wat betekent: $P_i$ is toepasbaar in situatie S

De volgende definities hebben betrekking op meer specifieke vormen van beslistabellen:

6. **Limited input DT:** alleen binaire condities en acties zijn toegestaan

7. **Extended input DT:** multi-valued condities en acties zijn toegestaan

[*] *Deze definitie is afkomstig uit het werk van Eremeev [8]. Hierin waren echter de symbolen ∀ en ∃ verkeerd om gebruikt. De definitie is hier gecorrigeerd.*

Een extended input DT is overigens altijd om te zetten in en limited input DT en omgekeerd. Zie het voorbeeld van de tabellen 4 en 5.

| Naam | Mark | Onno | Rineke |
|---|---|---|---|
| Functie | Kennistechnoloog | Technisch manager | UHD |

**Tabel 4: Extended input DT**

| | | | |
|---|---|---|---|
| Naam is Mark | Ja | Nee | Nee |
| Naam is Onno | Nee | Ja | Nee |
| Naam is Rineke | Nee | Nee | Ja |
| Functie Kennistechnoloog | Ja | Nee | Nee |
| Functie Technisch manager | Nee | Ja | Nee |
| Functie UHD | Nee | Nee | Ja |

**Tabel 5: Limited input DT**

8. **Enkelvoudige regel:** Een regel waarvan de conditiekolom geen * of [] bevat.

9. **Gegeneraliseerde regel:** Een regel waarvan de conditiekolom een * of een [] bevat.

10. **Expanded DT (Canonical DT):** Een DT die nergens een * of een [] in de condities bevat, oftewel, die alleen enkelvoudige regels bevat.

11. **Contracted DT (Consolidated DT):** Een DT die een * of een [] in de condities bevat, oftewel, die gegeneraliseerde regels bevat

12. **Limiting DT:** Een DT die geen identieke actiekolommen bevat.

De voorbeelden van Tabellen 6, 7 en 8, gebaseerd op voorbeelden uit [21] verduidelijken de begrippen die zojuist zijn genoemd. Bovendien is hier te zien dat een contracted DT altijd is om te schrijven naar een expanded DT.

| Conditieonderwerp 1 | A | | | B | | | C | | |
|---|---|---|---|---|---|---|---|---|---|
| Conditieonderwerp 2 | X | Y | Z | X | Y | Z | X | Y | Z |
| Actieonderwerp 1 | P | Q | R | S | T | T | U | U | U |

**Tabel 6: Expanded DT: bevat alleen enkelvoudige regels**

| Conditieonderwerp 1 | A | | | B | | | C |
|---|---|---|---|---|---|---|---|
| Conditieonderwerp 2 | X | Y | Z | X | Y | Z | * |
| Actieonderwerp 1 | P | Q | R | S | T | T | U |

**Tabel 7: Contracted DT: bevat gegeneraliseerde regels**

| Conditieonderwerp 1 | A | | | B | | C |
|---|---|---|---|---|---|---|
| Conditieonderwerp 2 | X | Y | Z | X | [] | * |
| Actieonderwerp 1 | P | Q | R | S | T | U |

**Tabel 8: Contracted en Limiting DT: bevat gegeneraliseerde regels en geen identieke actiekolommen.**

13. **Volledige DT:** Een DT waarbij elke conditiecombinatie $CK_j$ een actieconfiguratie $AK_j$ kent.

14. **Exclusieve DT:** Een DT waarbij elke conditiecombinatie $CK_j$ slechts één actieconfiguratie $AK_j$ kent.

15. **Mutually exclusive DT (single hit DT):** Een volledige en exclusieve DT.

De Tabellen 6,7 en 8 zijn allen mutually exclusive DT's. Alle conditieconfiguraties kennen één en slechts één actieconfiguratie.

### 2.1.3 Systemen met zowel regels als beslistabellen

Voor systemen met zowel regels als beslistabellen gelden grotendeels alle definities als die reeds genoemd zijn voor regelgebaseerde systemen en beslistabelgebaseerde systemen. Het enige verschil is dat de KB nu bestaat uit een verzameling regels (Rule Base) plus een verzameling beslistabellen (DT Base) en een declaratieset (Declaration Set):

1. **KB** = Rule Base ∪ DT Base ∪ Declaration Set

De business logica BL is de vereniging van de Rule Base en de DT Base:

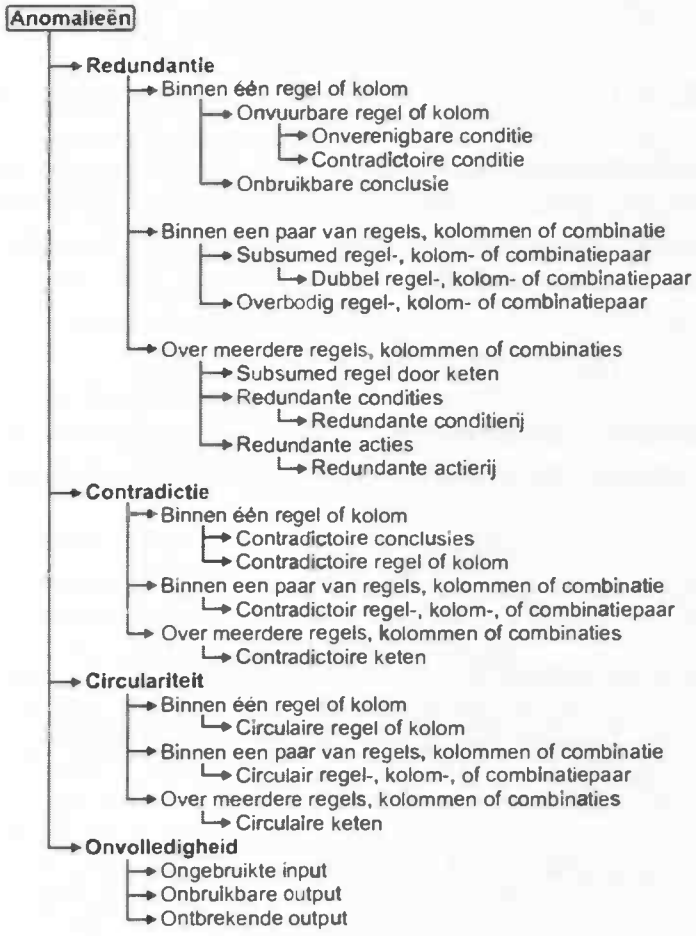
2. **BL** = Rule Base ∪ DT Base

## 2.2 Beschrijving van anomalieën

Zoals in hoofdstuk 1 reeds is vermeld kunnen er verschillende soorten van anomalieën optreden in een KB. De hoofdklassen waaronder deze kunnen worden ingedeeld zijn: redundantie, contradictie, circulariteit en onvolledigheid. In de literatuur wordt contradictie overigens ook wel *conflict* of *ambivalentie* [2] genoemd en wordt onvolledigheid ook wel *deficiëntie* genoemd.
Anomalieën kunnen voorkomen in regelgebaseerde systemen, beslistabellen en in gecombineerde systemen. In de volgende paragrafen zal een overzicht worden gegeven met formele definities van alle categorieën en subcategorieën die mogelijk zijn.

### 2.2.1 Categorisering

Preece heeft in [2] de basis gelegd voor de categorisering in regelgebaseerde systemen. Du Zang en Luqi hebben in [9] deze verder uitgewerkt en een aantal categorieën toegevoegd. Vanthienen en anderen [5, 6, 7] hebben met name gewerkt aan de categorisering van anomalieën in beslistabellen, zowel binnen één beslistabel als anomalieën die optreden in systemen met meerdere beslistabellen. Het bleek dat de categorieseringen veel overeenkomsten hadden en samengevoegd konden worden in één schema. Dit schema is te zien in Figuur 4.




**Figuur 4: Categorisering van anomalieën**

### 2.2.2 Definities

Hieronder volgt een beschrijving met formele definities van de verschillende categorieën, tevens op basis van het werk van Preece [2], Zhang & Luqi [9] en Vanthienen [6, 7]. Er wordt voortgebouwd op de definities in paragraaf 2.1. Wanneer in deze definities wordt gesproken over 'conclusies' kunnen zowel de conclusies van regels bedoeld worden als de acties van een kolom in een tabel. Van elke hoofdcategorie wordt eerst een algemene definitie gegeven waarna specifieke definities volgen voor de diverse subcategorieën. De definities voor de subcategorieën zijn zoveel mogelijk dusdanig geformuleerd dat deze elkaar uitsluiten (ze zijn exclusief)*. Voor extra begrip wordt de lezer verwezen naar Appendix B waarin voorbeelden zijn opgenomen van alle subcategorieën.

1. **Redundantie (algemene definitie):** Een regel of kolom R is redundant wanneer voor elke environment geldt dat de afleidbare hypotheses van de Business Logic **BL** gelijk zijn, onafhankelijk van de aanwezigheid of afwezigheid van R.

   $(\forall E \in \mathbf{E})(\{H \mid infer(H, \mathbf{BL}, E)\} = \{H \mid infer(H, \mathbf{BL} - \{R\}, E)\})$

2. **Redundantie: onvuurbare regel of kolom:** Een regel of kolom R is onvuurbaar wanneer er geen enkele environment E bestaat waarbij de condities van R een logisch gevolg zijn van E toegepast op Business Logic **BL**.

   $\neg((\exists E \in \mathbf{E})\ (firable(R, \mathbf{BL}, E))$

3. **Redundantie: onverenigbare condities:** Een regel of kolom R bevat onverenigbare condities wanneer minstens één van de condities niet behoort tot de set van input literals en tevens geen deel uit maakt van de conclusies van een andere regel of kolom in de Business Logic **BL**.

   $(\exists L \in condit(R))\ (L \notin \mathbf{I}) \wedge \neg\ (\exists R' \in (\mathbf{BL} - \{R\}))(L \in concl(R'))$

4. **Redundantie: contradictoire condities:** Een regel of kolom R bevat contradictoire condities wanneer de condities leiden tot een semantische of syntactische contradictie.

   $(\exists C \in \mathbf{C})\ (condit(R) \vdash C) \vee (condit(R) \vdash \perp)$

5. **Redundantie: onbruikbare conclusies:** Een regel of kolom R bevat onbruikbare conclusies wanneer minstens één van de conclusies niet behoort tot de set van goal literals en tevens geen deel uit maakt van de condities van een andere regel of kolom in de Business Logic **BL**.

   $(\exists L \in concl(R))\ (\ (L \notin \mathbf{G}) \wedge \neg(\exists R' \in (\mathbf{BL} - \{R\}))(L \in condit(R'))\ )$

---

* *Uitzonderingen hierop zijn definitie 2: onvuurbare regel of kolom, die een overlappende definitie vormt voor definities 3, 4 en 5 en definitie 7: dubbel regel-, kolom- of combinatiepaar, welke een speciaal geval is van definitie 6.*

6. **Redundantie: subsumed regel-, kolom- of combinatiepaar:** Wanneer de condities van een regel of kolom R een subset vormen van de condities van een regel of kolom R', en de conclusies van R' vormen een subset van de conclusies van R, dan is R' een overbodige regel of kolom.

   $(\exists R \in \mathbf{BL}, \exists R' \in \mathbf{BL})\ (\ (antec(R) \subseteq antec(R')) \wedge (concl(R') \subseteq concl(R))\ )$

7. **Redundantie: dubbel regel-, kolom- of combinatiepaar:** Wanneer de Business Logic tweemaal hetzelfde regel-, kolom- of combinatiepaar bevat, dan is één van de twee redundant. (een speciaal geval van subsumed regel-, kolom- of combinatiepaar).

   $(\exists R \in \mathbf{BL}, \exists R' \in \mathbf{BL})\ ((condit(R) = condit(R') \wedge (concl(R') = concl(R))$

8. **Redundantie: overbodig regel-, kolom- of combinatiepaar:** Wanneer een regel of kolom R en een andere regel of kolom R' gelijke conclusies hebben en R bevat slechts één conditieliteral L en R' bevat slechts één conditieliteral $\neg$L, dan zijn R en R' beide redundant wanneer de conclusie wordt toegevoegd aan de inputverzameling.

   $(concl(R) = concl(R')) \wedge (condit(R) = (L)) \wedge (condit(R') = (\neg L))$

9. **Redundantie: subsumed regel of kolom door keten:** Wanneer de conclusies van een regel of kolom $R_x$ uit de **BL** in ieder mogelijk environment kunnen worden afgeleid via een set van andere regels en/of kolommen$\{R_1, \ldots, R_n\}$ uit de **BL**, en bovendien is $R_x$ niet onvuurbaar, bevat geen onbruikbare conclusies, is niet subsumed door een andere regel en behoort niet tot een overbodig paar, dan is $R_x$ subsumed door een keten. $R_x$ voldoet dan aan de algemene definitie voor redundantie, definitie 1, maar aan geen enkele van de definities 2 tot en met 8.

10. **Redundantie: redundante conditierij:** Wanneer een DT een conditieonderwerp bevat waarvan alle conditiewaarden irrelevant zijn, dan is het gehele conditieonderwerp, en daarmee de gehele rij, redundant.

11. **Redundantie: redundante conclusierij:** Wanneer er twee rijen zijn in het conclusiegedeelte van een DT, A en A' met identieke conclusieonderwerpen en de conclusiewaarden van A vormen een subset van de conclusiewaarden van A', dan is de gehele conclusierij A redundant.

12. **Contradictie (algemene definitie):** De Business Logic **BL** bevat contradictoire regels wanneer voor een mogelijke environment E alle literals van een semantische contradictie C kunnen worden afgeleid of wanneer een syntactische contradictie kan worden afgeleid.

    $(\exists E \in \mathbf{E}, \exists C \in \mathbf{C})\ (((\forall L \in C)\ infer(L, \mathbf{R}, E)) \vee infer(\perp, \mathbf{R}, E))$

13. **Contradictie: contradictoire conclusies:** Wanneer een regel of kolom R conclusies bevat die een syntactische of semantische contradictie C vormen, dan bevat deze regel contradictoire conclusies.

$(\exists C \in \mathbf{C})\ (\text{concl}(R) \vdash C) \vee (\text{concl}(R) \vdash \bot)$

14. **Contradictie: contradictoire regel of kolom:** Een regel of kolom R is contradictoir wanneer de condities verenigd met de conclusies van R een syntactische of semantische contradictie C vormen en deze contradictie is niet afleidbaar uit slechts de condities van R of slechts de conclusies van R.

$(\exists C \in \mathbf{C})\ (\ (\ (\text{condit}(R) \cup \text{concl}(R) \vdash C) \vee (\text{condit}(R) \cup \text{concl}(R) \vdash \bot)\ )$
$\wedge\ \text{condit}(R) \nvdash C \wedge \text{condit}(R) \nvdash \bot \wedge \text{concl}(R) \nvdash C \wedge \text{concl}(R) \nvdash \bot\ )$

15. **Contradictie: contradictoir regel-, kolom- of combinatiepaar:** De Business Logic **BL** bevat een contradictoir regel-, kolom- of combinatiepaar R en R' wanneer aan één van de twee volgende voorwaarden* is voldaan:

a) De condities van R vormen een subset van de condities van R' en de conclusies van R verenigd met de conclusies van R' vormen een semantische contradictie C of een syntactische contradictie.

$(\exists C \in \mathbf{C}, \exists R \in \mathbf{BL}, \exists R' \in \mathbf{BL})\ (\ (\text{condit}(R) \subseteq \text{condit}(R')) \wedge$
$(\ ((\text{concl}(R) \cup \text{concl}(R')) \vdash C) \vee ((\text{concl}(R) \cup \text{concl}(R')) \vdash \bot)\ )\ )$

b) De condities van R' vormen een subset van de conclusies van R en de condities van R verenigd met de conclusies van R en verenigd met de conclusies van R' vormen een semantische contradictie C of een syntactische contradictie.

$(\exists C \in \mathbf{C}, \exists R \in \mathbf{BL}, \exists R' \in \mathbf{BL})\ (\ (\text{condit}(R') \subseteq \text{concl}(R)) \wedge$
$(\ ((\text{condit}(R) \cup \text{concl}(R) \cup \text{concl}(R')) \vdash C) \vee$
$((\text{condit}(R) \cup \text{concl}(R) \cup \text{concl}(R')) \vdash \bot)\ )\ )$

Bovendien moet gelden dat de contradictie die kan worden afgeleid via één van bovenstaande voorwaarden, niet kan worden afgeleid volgens definities 13 of 14 toegepast op alleen R of R'.

16. **Contradictie: contradictoire keten:** De Business Logic **BL** bevat een contradictoire keten wanneer in een mogelijke environment E een set van regels $\{R_p, ..., R_q\}$ vuurbaar zijn die samen een contradictie vormen volgens de algemene definitie 12, en deze contradictie is niet afleidbaar volgens één van de andere definities voor contradictie: 13, 14 of 15.

---

* *In de definities van Preece en Shinghal [2] wordt de tweede mogelijkheid van een contradictoir regelpaar niet genoemd. Een dergelijke situatie is echter wel degelijk contradictoir. Wanneer het verschil tussen beide situaties niet geheel duidelijk is, verwijzen we de lezer graag naar de voorbeelden in Appendix B.*

17. **Circulariteit (algemene definitie):** De Business Logic **BL** bevat een circulariteit wanneer een regel of kolom R alleen vuurbaar is wanneer de conclusie van R als input wordt gegeven aan de **BL**. R is dan afhankelijk van zichzelf.

    $(\exists R \in \mathbf{BL}, \exists E \in \mathbf{E})\ (\neg firable(R, \mathbf{BL}, E) \wedge firable(R, \mathbf{BL}, E \cup \{concl(R)\})$

18. **Circulariteit: Circulaire kolom of regel:** Een regel of kolom R waarbij één van de condities van R tevens één van de conclusies vormt van R.

    $\exists L\ (L \in condit(R) \wedge L \in concl(R))$

19. **Circulariteit: Circulair regel-, kolom- of combinatiepaar:** Wanneer de condities van een regel of kolom R het literal L bevatten en de conclusies van R bevatten L' terwijl de condities van een andere regel of kolom R' L' bevatten en de conclusies van R' bevatten L, dan is het paar circulair.

    $\exists L, \exists L'\ (L \in condit(R) \wedge L' \in concl(R) \wedge L' \in condit(R') \wedge L \in concl(R'))$

20. **Circulariteit: Circulaire keten:** Een KB bevat een circulaire keten van regels $\{R_p, \ldots, R_q\}$ wanneer deze keten voldoet aan de algemene definitie van circulariteit, definitie 17, en deze circulariteit is niet af te leiden via de definities 18 en 19.

21. **Onvolledigheid: Onbruikbare output**[*]**:** Er is sprake van onbruikbare output wanneer er een environment E bestaat waarbij geen enkele goal uit de Goal Set kan worden afgeleid uit de Business Logica.

    $(\exists E \in \mathbf{E})\ ((\forall G \in \mathbf{G})\ (G \notin \{H \mid infer(H, \mathbf{BL}, E)\}))$

22. **Onvolledigheid: Ontbrekende output**[#]**:** Er is sprake van ontbrekende output wanneer er een environment E bestaat waarbij niet alle gewenste goals op basis van environment E kunnen worden afgeleid uit de Business Logica.

    $(\exists E \in \mathbf{E})\ ((\exists G \in \mathbf{G_E})\ (G \notin \{H \mid infer(H, \mathbf{BL}, E)\}))$

    met: $\mathbf{G_E}$ is een subset van **G** welke bestaat uit de gewenste goals die afleidbaar zouden moeten zijn uit environment E, via **BL**.

---

[*] *De definitie die hier wordt gegeven van onbruikbare output is vrijwel gelijk aan de definitie van* ontbrekende regels *van Preece & Shinghal [2]. Logischerwijs kan inderdaad gesteld worden dat er regels ontbreken (of uitgebreid moeten worden) wanneer er geen enkele bruikbare goal kan worden afgeleid.*

[#] *De definitie van ontbrekende output is een toevoeging die niet wordt vermeld in [2]. Men kan zich echter voorstellen dat de gebruiker van te voren vaststelt wat de set van gewenste goals is voor een bepaald environment E. Als niet alle gewenste goals worden afgeleid, is sprake van onvolledigheid. Ontbrekende output kan worden gezien als een specifieke vorm van onbruikbare output.*

23. **Onvolledigheid: Ongebruikte input** = Business Logica bevat ongebruikte input wanneer er een input literal I bestaat die niet wordt gebruikt; het is geen goal en het wordt bij geen enkele regel of kolom als conditie gebruikt.

$(\exists I \in \mathbf{I})\ ((I \notin \mathbf{G}) \wedge \neg(\exists R \in \mathbf{BL})\ (I \in \text{condit}(R)))$

## 2.3 Detectie van anomalieën

### 2.3.1 Principes

De automatische detectie van anomalieën kan plaatsvinden in een 'stand alone' applicatie of in een geïntegreerde module van de ontwerpomgeving van kennissystemen. Een geïntegreerde module biedt als voordeel dat verificatie wellicht makkelijker en sneller toe te passen is tijdens het ontwerpproces. Een stand alone biedt als voordeel dat verificatie mogelijk is op meerdere verschillende systemen in eventueel andere talen en ontwikkeld in andere omgevingen. In die situatie zal dan wel een speciale vertaalmodule noodzakelijk zijn om de KB in de juiste representatie om te zetten. In beide gevallen gelden echter de volgende verificatie principes [2]:

1. Anomaliedetectie vindt alleen plaats binnen de KB. Idealiter is de detectie onafhankelijk van de inference engine.

2. Anomalieën worden gedefinieerd in termen van de declaratieve betekenis. Het gaat dus om de statische, en niet de procedurele betekenis.

3. Anomalieën worden gedetecteerd op basis van de syntaxis en op basis van de semantiek*.

4. De syntaxis en semantiek van de anomalieën moet in dezelfde vorm zijn als de syntaxis en semantiek van de KB.

5. Anomalieën zijn geen fouten: het zijn mogelijke fouten. Dit kwam in hoofdstuk 1 reeds ter sprake. Bepaalde vormen van redundantie, contradictie en onvolledigheid kunnen bewust zo zijn bedoeld door de ontwerpers. Soms kunnen anomalieën ook bewust in stand gehouden worden omdat men weet dat een toekomstige uitbreiding of wijziging van de KB de anomalieën zal verwijderen.

---

* *Volgens Preece [2] is het zeer moeilijk om anomalieën te detecteren op basis van semantiek. Ik ben het daar niet helemaal mee eens. In zijn eigen definities noemt hij al het opnemen van semantische contradicties in de declaratieset. Die semantische contradicties kunnen gebruikt worden om semantische anomalieën op te sporen.*

### 2.3.2 Methoden

De methoden waarop anomalieën kunnen worden gedetecteerd zijn globaal in drie categorieën te verdelen [2].

1. **Integriteittests:**
   Deze tests kenmerken zich door vergelijkingen van alle condities en conclusies van elke regel en kolom in de KB met de input set, de goal set, de set van semantische contradicties en de hypothese set. Toepassingen zijn: onverenigbare condities, contradictoire condities, onbruikbare conclusies, circulariteit en contradictie binnen één regel en ongebruikte input. Het detecteren van deze anomalieën gaat feitelijk via twee stappen. Binnen een regel en kolom moeten alle condities en conclusies vergeleken worden met elkaar of met eventueel de input set, de goal set, de set van semantische contradicties en de hypothese set. De complexiteit van deze stap neemt exponentieel toe met het aantal literals dat met elkaar moet worden vergeleken (bijvoorbeeld: kwadratisch voor het vergelijken van alle mogelijke paren van literals). De tweede stap is, dat deze tests gedaan moeten worden voor alle regels en kolommen in de KB. Dit kan echter serieel, dus per individuele regel en kolom, gedaan worden. De theoretische computationele complexiteit voor integriteittests kan daarom ook opgebouwd worden uit twee componenten: Een bepaalde vaste (gemiddelde) complexiteit voor het vergelijken van de literals (binnen een regel of kolom), en een totale complexiteit (voor de test over alle regels en kolommen) die lineair toeneemt (N) met het aantal regels en kolommen in de KB.

2. **Regeltests:**
   Deze tests kenmerken zich door een algoritme dat alle regels en kolommen (condities en conclusies) vergelijkt met alle andere regels en kolommen van de KB. Toepassingen zijn: subsumed of dubbel regel-, kolom- of combinatiepaar, overbodig regel-, kolom- of combinatiepaar en circulariteit of contradictie binnen een regel-, kolom- of combinatiepaar. De theoretische computationele complexiteit voor regeltests is net als die van integriteittests opgebouwd uit twee componenten: Een bepaalde vaste (gemiddelde) complexiteit voor het vergelijken van de literals (binnen een regel of kolom), en een totale complexiteit (voor de test over alle regels en kolommen) die ditmaal kwadratisch toeneemt ($N^2$) met het aantal regels en kolommen in de KB.

3. **Regelextensietests:**
   Hierbij is het in principe noodzakelijk dat de gehele extensie van de KB wordt berekend: elke mogelijke afleidingsketen op basis van alle mogelijke environments moet worden doorlopen. Computationeel kan dit zeer veeleisend zijn. Volgens Preece [2] is de hoeveelheid paden die wordt berekend $b^d$, waarbij b de breedte is van de KB (het gemiddeld aantal literals in de condities) en d is de diepte van de KB (het gemiddeld aantal regels en/of kolommen in een afleidingsketen). Toepassingen zijn: subsumed regel of kolom door een keten en contradictie en circulariteit in een keten, ongebruikte en onbruikbare output.

Wat betreft de complexiteit van integriteittests en de regeltests kan de volgende opmerking gemaakt worden. Zoals aangegeven is complexiteit voor deze tests afhankelijk van de complexiteit van de eerste component; het vergelijken van de

literals. Echter, in de praktijk bedraagt de gemiddelde lengte van de regels en kolommen uit een standaard KB nooit meer dan 7 of 8 literals. Deze eerste component kan daarom gezien worden als een constante, en de totale complexiteit zal vooral afhangen van het aantal regels in de KB.

Ten aanzien van de complexiteit van regeltests kan worden opgemerkt dat in bepaalde situaties voorselecties bij het testen ervoor kunnen zorgen dat niet altijd alle regels en kolommen met alle andere regels en kolommen vergeleken hoeven te worden. De computationele complexiteit is dus niet precies $N^2$ maar maximaal $N^2$. Ook wat betreft de complexiteit van regelextensietests geldt dat deze niet precies, maar maximaal $b^d$ bedraagt.

Wanneer sprake is van een grote hoeveelheid regels en beslistabellen die een hoge mate van onderlinge afhankelijkheid bezitten, kan het detectieproces veel tijd in beslag nemen. Soms moet een afweging worden gemaakt tussen het detecteren van zoveel mogelijk anomalieën en de snelheid van het detectieproces. Het toepassen van heuristieken en slimme voorselecties die slechts een gedeeltelijke extensie van alle inference paden afleiden, kan uitkomst bieden.

In hoofdstuk 6, Model & Algoritmen, zal duidelijk worden dat deze algemene theoretische beschrijvingen van detectiemethoden, ook daadwerkelijk een basis vormen voor de algoritmiek en de prototype verificatietool die zijn geconstrueerd. Eerst volgt nu hoofdstuk 3 waarin enige reeds bestaande verificatietools worden geanalyseerd.

# H3 ANALYSE VAN ENKELE BESTAANDE TOOLS

In dit hoofdstuk zullen een aantal reeds bestaande tools worden besproken die in staat zijn om anomalieën in een kennissysteem te detecteren. Het doel hiervan is ten eerste vast te stellen of deze tools wellicht reeds dusdanig compleet zijn dat deze zonder veel problemen ingezet zouden kunnen in het Knowledge Framework van Everest. Er zal blijken dat dit niet het geval is. Ten tweede verschaft dit hoofdstuk meer inzicht in verificatie in de praktijk en de problemen die daarbij aan de orde zijn.

Er is een redelijk aantal tools beschikbaar voor de validatie en verificatie van kennissystemen. S. Murell en R. Plant hebben een literatuurstudie gedaan over de publicaties over validatie- en verificatietools voor kennissystemen in de periode 1985-1995 [16]. Er werden in totaal 33 tools onderzocht, vrijwel allemaal afkomstig uit de academische wereld. Eén van de vele onderzochte karakteristieken was het testen op anomalieën via formele of semi-formele technieken, met andere woorden, het onderdeel verificatie. Slechts 11 van de 33 tools beschikten over dergelijke technieken. De schrijvers Murell en Plant stellen dat dit lage aantal berust op het feit dat er nog steeds onvoldoende fundamenteel onderzoek is verricht is op het gebied van formele technieken voor anomaliedetectie. De tools die anomaliedetectie ondersteunden waren onder andere: VERITE, CONKRET, PROLOGA, CLINT, COVER, SACCO, P/HUNTER en KB Reducer3. De periode na 1995 viel niet meer binnen de studie van Murell en Plant. Een eigen literatuuronderzoek over de periode na 1995 resulteerde in nog een tweetal interessante tools die anomaliedetectie ondersteunen: VALENS en VERITAS.

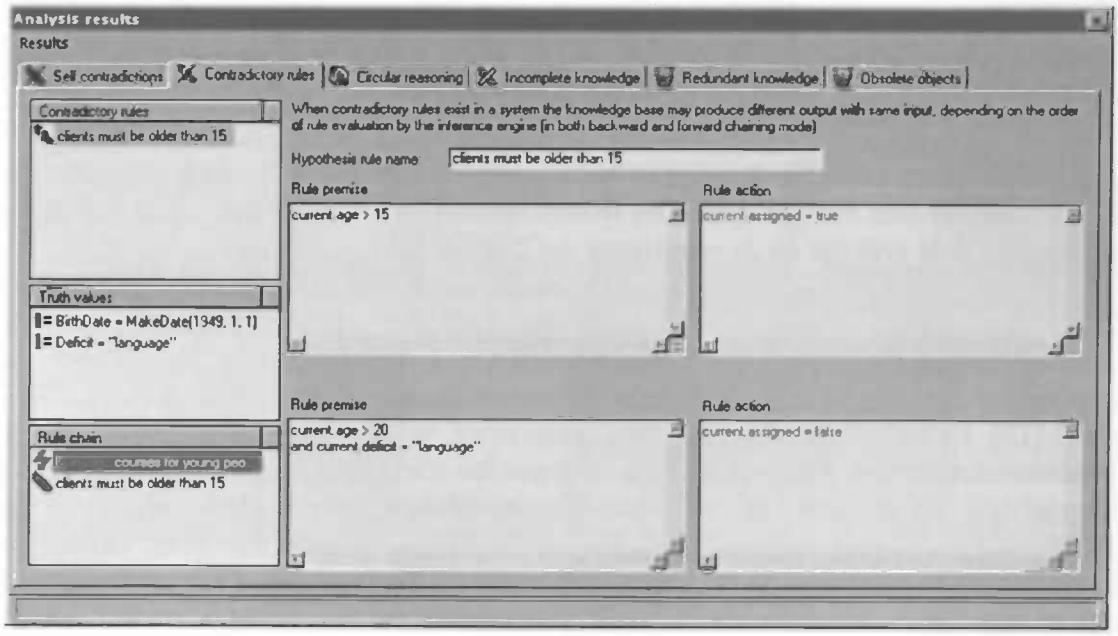
In de volgende paragrafen van dit hoofdstuk zal de werking en functionaliteit van VALENS, PROLOGA, COVER en VERITAS worden besproken. De keuze om deze tools te analyseren en andere niet, heeft vooral te maken met de beschikbaarheid van literatuur over de tool en/of de beschikbaarheid van de tool zelf.

## 3.1 VALENS

VALENS is een van de weinige commerciële tools die beschikbaar zijn voor de validatie en verificatie van een Knowledge Base. Het was niet haalbaar om deze tool aan te schaffen alleen voor dit afstudeeronderzoek. De analyse van VALENS zal daarom alleen gebaseerd op een aantal artikelen [11, 12, 13, 14] van de ontwikkelaars van de tool, S. Spreeuwenberg en R. Gerrits.

VALENS is speciaal ontwikkeld voor de validatie en verificatie van een Aion Knowledge Base. Aion is een objectgeoriënteerde programmeeromgeving voor kennissystemen. In principe worden alleen Aion 8 KB's geaccepteerd, hoewel in bepaalde gevallen andere KB's geconverteerd kunnen worden. VALENS is ontwikkeld als een extra module of component van Aion, en dus niet als een stand alone. Zowel beslistabellen als regels worden ondersteund. De KB wordt door VALENS getest op alle hoofdcategorieën; consistentheid, circulariteit, compleetheid en redundantie. De output van VALENS is een document waarin alle ongeldige regels

of ketens van regels worden aangegeven met een classificatie en uitleg. Een voorbeeld van de output (overgenomen uit [11]) is te zien in Figuur 5.




**Figuur 5: Resultaten window van VALENS**

VALENS gaat in drie stappen te werk:

1. Constructie van een metamodel. Dit metamodel bevat de vooraf gedefinieerde mogelijke input- en outputwaarden van het systeem en maximum- en minimumwaarden van variabelen en attributen. Het metamodel kan hier dus worden gezien als de declaratieset. Tevens worden de regels in de KB gescheiden in condities en conclusies. De condities worden vervolgens weer gescheiden in clauses (kennelijk zijn meerdere clauses of literals in de conclusie niet mogelijk in VALENS, hoewel dit in geen van de artikelen expliciet gemeld wordt).

2. Selectie van kandidaten voor anomalieën. Er is een selectie van potentiële kandidaten via het gebruik van heuristieken of er kan een directe aanwijzing van anomalieën zijn.

3. Bewijs van de anomalieën. Via de inference engine worden de mogelijke kandidaten getest, via forward chaining. Dezelfde inference engine die normaalgesproken de regels van de KB evalueert en vuurt wordt dus gebruikt om anomalieën te detecteren. Deze techniek wordt *proof-by-processing* genoemd.

VALENS kan omgaan met een rijkere taal dan de propositielogica, gebaseerd om de predikatenlogica. Hierdoor kunnen functies worden gebruikt (en ingevuld) tijdens het testen (bijvoorbeeld de functie 'Leeftijd'). Verder kan gebruik gemaakt worden van een objectgeoriënteerde omgeving, waarbij attributen kunnen worden overgeërfd. De techniek van proof-by-processing zorgt ervoor dat er geen verschil is tussen run-time logica en de logica die gebruikt wordt in het verificatieproces, aangezien dezelfde inference engine wordt gebruikt.

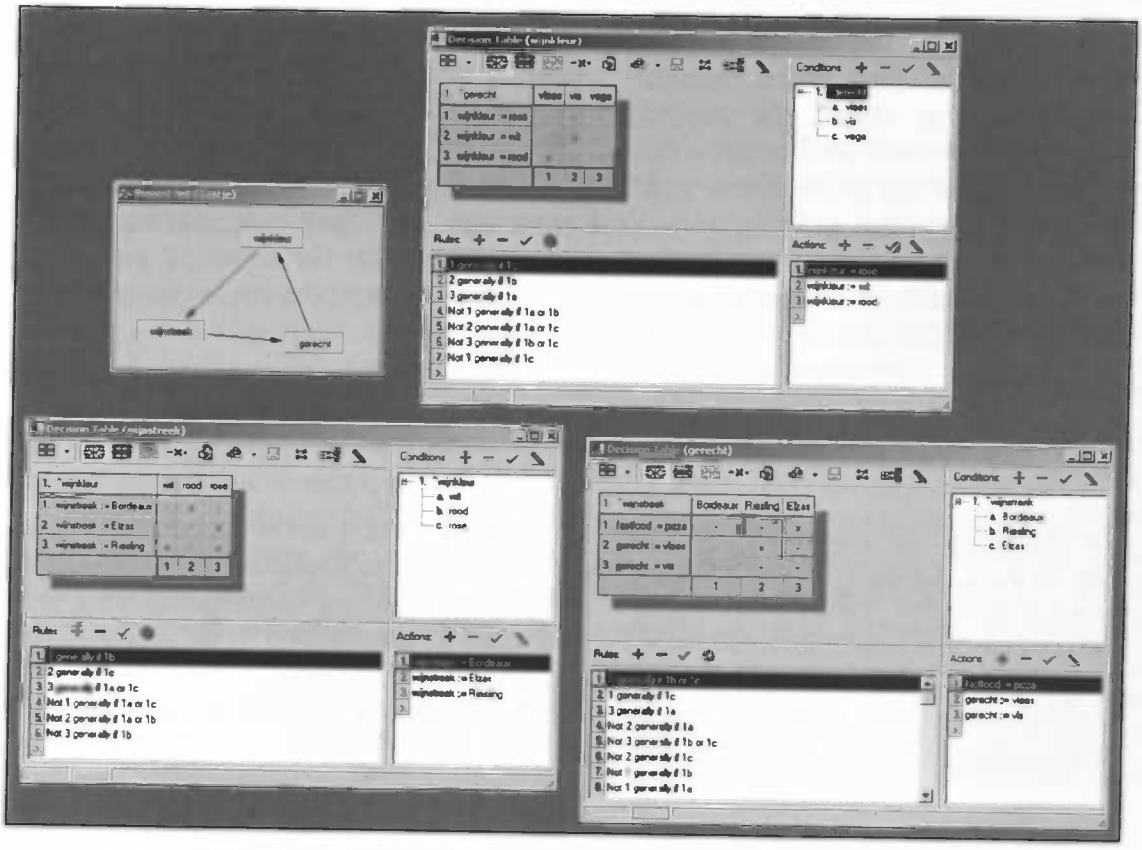
Een belangrijk nadeel is dat VALENS geen semantische contradicties (constraints) kan detecteren. Een ander mogelijk nadeel is het feit dat voor de selectie van kandidaten van regels die mogelijk anomalieën bevatten heuristieken worden gebruikt. Er wordt dus geen volledige regelextensie opgebouwd door VALENS (waarschijnlijk om performance redenen). Het is dus mogelijk dat hierdoor niet alle anomalieën worden gedetecteerd. Veel meer kan hier niet over worden gezegd aangezien de precieze werking van de heuristieken in de artikelen niet wordt besproken. Dit is overigens niet verwonderlijk met betrekking tot het commerciële karakter van VALENS.

## 3.2 PROLOGA

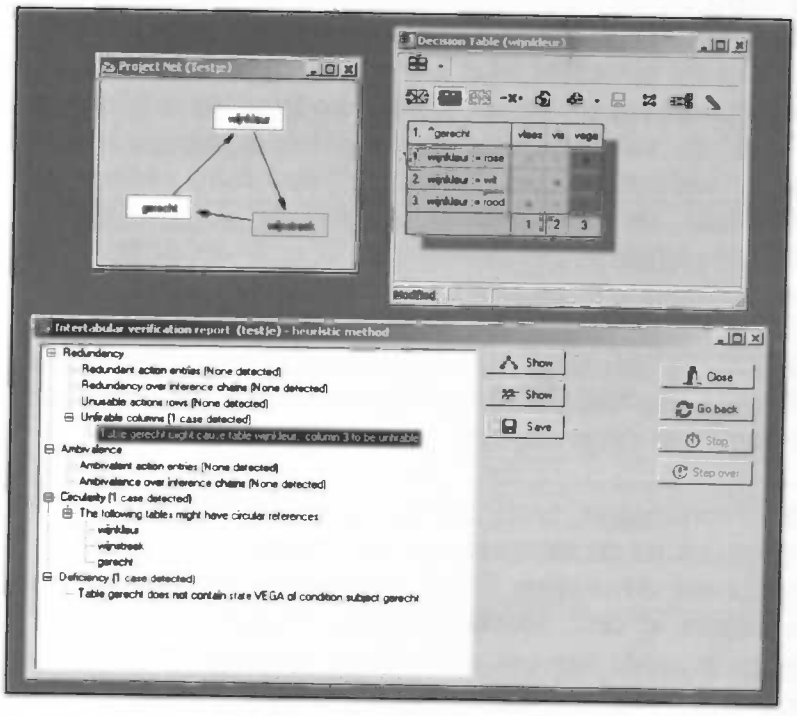
PROLOGA (PROcedural LOGic Analyzer) [5, 6, 7] is ontwikkeld door J. Vanthienen aan de K.U. Leuven en onderscheidt zich van de andere tools die in dit hoofdstuk worden genoemd. PROLOGA vormt namelijk een volledige ontwerpomgeving voor het creëren van beslistabelsystemen. De detectie van anomalieën in de beslistabellen is een geïntegreerd onderdeel van PROLOGA en kan daarom gemakkelijk tijdens de ontwerpfase worden toegepast (incrementele verificatie). Het programma is getest en de bevindingen volgen hieronder.

PROLOGA ondersteunt alleen beslistabellen. De beslistabellen in PROLOGA zijn limited in het actiegedeelte (hier zijn dus alleen binaire waarden toegestaan). De beslistabellen worden gecreëerd als volledig expanded tabellen en moeten mutually exclusive zijn. Deze werkwijze zorgt tijdens de creatie van een tabel direct al dat er geen redundantie of onvolledigheid binnen de tabel kan zijn. Nadat de tabel correct is bevonden, kan de tabel gesimplificeerd worden door deze om te zetten in contracted formaat. Eventueel kan ook de volgorde van de condities aangepast worden. PROLOGA beschikt over technieken om zowel intratabulaire anomalieën (anomalieën binnen één tabel) als intertabulaire anomalieën (anomalieën die plaatsvinden binnen meerdere tabellen) te detecteren. Vooral de detectie van intertabulaire anomalieën is belangrijk, aangezien anomalieën binnen een beslistabel waarschijnlijk minder vaak zullen voorkomen door de werkwijze die zojuist besproken is. De technieken om intertabulaire anomalieën te detecteren maken gebruik van heuristieken waar een uitputtende test te inefficiënt zou zijn. Een schermvoorbeeld van de PROLOGA omgeving is te zien in Figuur 6.

In dit voorbeeld zijn drie eenvoudige beslistabellen gecreëerd die samen een circulariteit vormen. Dit volgt ook uit de weergave van het 'Project Net'. Daarnaast is er een redundante actierij en een onvuurbare kolom aanwezig. De rapportage van intertabulaire anomalieën melden al deze anomalieën zoals te zien is in Figuur 7. Tevens wordt er een melding gemaakt van een onvolledige (deficiënt) actierij in de tabel 'wijnkleur'. Deze classificatie is enigszins verwarrend. Classificatie als onbruikbare actierij was in dit geval beter geweest.



**Figuur 6: Schermvoorbeeld van de PROLOGA omgeving**



**Figuur 7: PROLOGA verificatierapportage**

Over het algemeen functioneert PROLOGA goed, maar er zijn ook een aantal nadelen aan te wijzen. Ten eerste is er geen Goal Set en geen Input Set. Bij een consultatie van de beslistabellen wordt alleen een starttabel gekozen en op basis hiervan worden alle afleidingen gemaakt die mogelijk zijn. De afwezigheid van een Goal Set en een Input Set resulteert in het feit dat bepaalde categorieën van redundantie en onvolledigheid

niet kunnen worden gedetecteerd, zoals onverenigbare condities, onbruikbare conclusies en onbruikbare en ontbrekende output. Daarnaast zijn attributen in PROLOGA waarschijnlijk altijd multivalued. Dit blijkt bijvoorbeeld uit de rapportage in Figuur 7. Als de attributen singlevalued waren, dan zouden er wel degelijk contradicties gemeld moeten worden. Verder is het (net als in VALENS) niet mogelijk om semantische contradicties te declareren.

Een handige functionaliteit van PROLOGA is de mogelijkheid een beslistabelsysteem om te zetten in programmacode voor verschillende programmeeromgevingen zoals Aion, Delphi of C++. Dit maakt PROLOGA in principe een multiplatformsysteem.

## 3.3 COVER

COVER is een stand-alone tool die speciaal ontwikkeld is voor de verificatie van kennissystemen gebaseerd op regels. COVER is ontwikkeld door A. Preece. De tool en de prestaties zullen hier kort beschreven worden op basis van een artikel van A. Preece [2]. Ten einde anomalieën te detecteren voert COVER integriteittests, regeltests en extensietests uit (zie paragraaf 2.3.2). Tevens is er de mogelijkheid om semantische contradicties te declareren. In de outputvoorbeelden uit het artikel zijn alleen literals in de vorm (attribuut = waarde) te zien. Het is daarom niet waarschijnlijk dat de relaties <, >, <= en >= worden ondersteund, evenals negatieve literals.

COVER heeft geen aparte tests voor circulaire regels, contradictoire conclusies, circulair en contradictoir regelpaar. Hoewel Preece in zijn artikel de basis schept voor de categorisering van anomalieën, komen dus niet alle verschillende categorieën uit het artikel tot uiting in COVER. Echter, aangezien er wel algemene tests zijn voor contradictie en circulariteit (de tests voor circulaire en contradictoire ketens) zullen deze anomalieën waarschijnlijk wel worden gedetecteerd. Dit houdt echter in dat er computationeel veel complexere tests dan noodzakelijk moeten worden uitgevoerd om deze anomalieën te detecteren. Dit is een onhandig aspect van COVER. Het is overigens niet duidelijk uit het artikel of de categorieën die wel ondersteund worden überhaupt los van elkaar kunnen worden getest.

COVER is getest op vijf kennissystemen uit de praktijk met een Rule Base variërend van 105 tot 550 regels en een declaratieset variërend van 65 tot 510 declaraties. Drie van de vijf geteste systemen waren reeds geruime tijd in gebruik. Toch detecteerde COVER in al deze systemen een aanzienlijke hoeveelheid anomalieën (uiteenlopend van 15 tot meer dan 150 anomalieën in totaal), waarvan vele ook echte fouten bleken. Uitzonderingen waren redundante regels waarbij een specifiekere regel bewust aanwezig was naast een algemenere regel.

Interessant was dat integriteittests de meest effectieve tests bleken om anomalieën te ontdekken in termen van de hoeveelheid van deze anomalieën ten opzichte van de computationele kosten. Extensietests bleken alleen efficiënt bij kennissystemen waarbij de afleidketens gemiddeld redelijk lang zijn.

## 3.4 VERITAS

Hoewel de gevonden informatie over VERITAS wat beperkter was (slechts één bescheiden artikel [15] werd gevonden over deze tool) zal de tool hier toch kort behandeld worden. VERITAS is stand-alone verificatietool voor regelgebaseerde systemen. De tool is in principe domein- en grammaticaonafhankelijk, aangezien deze is uitgerust met een speciale converteermodule. In de praktijk zullen echter waarschijnlijk niet alle kennissystemen geschikt zijn voor de VERITAS tool, gezien de complexiteit van het omzetten van zeer verschillende grammatica's.

De werking van VERITAS gaat als volgt. Uit alle regels worden de literals geabstraheerd. Deze worden geclassificeerd als feiten (als ze alleen in de condities voorkomen), conclusies (als ze alleen als conclusie voorkomen) en hypotheses (als ze in beide zijden van de regels voorkomen). Verder kunnen er attributen met bijbehorende waarden worden gedeclareerd (de attributenset). VERITAS voert vervolgens een volledige extensie van alle regels uit. Alle mogelijke afleidketens worden dus geanalyseerd. VERITAS bevat tevens de mogelijkheid om semantische constraints te definiëren. Uit het artikel [15] bleek dat VERITAS problemen ondervond in het analyseren van KB's met variabelen met een range, bijvoorbeeld prijs- of tijdsintervallen. Waarom deze problemen precies ontstonden werd helaas niet vermeld.

## 3.5 Conclusies met betrekking tot de besproken tools

In Tabel 9 hieronder is een kort overzicht gegeven van de relevante eigenschappen van de vier besproken tools uit dit hoofdstuk.

| | VALENS | PROLOGA | COVER | VERITAS |
|---|---|---|---|---|
| **Representatie KB** | regels en tabellen | tabellen | regels | regels |
| **Omgeving** | alleen Aion 8 en 9 (hoewel conversie soms mogelijk) | PROLOGA is tevens de ontwerpomgeving maar conversie van KB code achteraf is mogelijk | stand alone: conversie van code vooraf | stand alone: conversie van code vooraf |
| **Syntax expressies** | propositionele deel predikatenlogica | propositionele deel predikatenlogica | propositionele deel predikatenlogica | propositionele deel predikatenlogica |
| **Mogelijke relaties** | =, <, >, >=, <= | = | = | =, <, >, >=, <= |
| **Negatieve literals** | ? | ja | nee | ? |
| **Sem. contradicties** | nee | nee | ja | ja |
| **Goal Set & Input Set** | ja | nee | ja | ? |
| **Single/multivalued attributen** | ? | alleen multivalued attributen | ? | alleen singlevalued attributen |
| **Heuristisch zoeken** | ja | ja | ja | nee |
| **Volledige extensies** | nee | gedeeltelijk | ja | ja |
| **Exclusief testen van alle categorieën** | nee | nee | nee | nee |
| **Categorieën die *niet* worden getest** | categorieën waar sem. contradicties een rol spelen, ontbrekende output | categorieën waar sem. contradicties een rol spelen, ongebruikte input, onbruikbare en ontbrekende output | overbodig regelpaar, ontbrekende output | ongebruikte input, onbruikbare en ontbrekende output |

**Tabel 9: Eigenschappen van VALENS, PROLOGA, COVER en VERITAS**

Wat opvalt is dat geen van de besproken tools volledig is met betrekking tot het testen van alle anomaliecategorieën zoals besproken in het vorige hoofdstuk. Wat betreft VALENS en PROLOGA is dit vooral te wijten aan het feit dat er geen semantische constraints kunnen worden gedeclareerd. Anomalieën die tot stand komen wegens semantische contradicties worden daarom niet gedetecteerd. Dit kan zijn in de categorieën: contradictoire condities, contradictoire conclusies, contradictoire regel, contradictoir regelpaar en contradictoire keten.

Een ander probleem is het feit dat het bij geen van de vier tools mogelijk is om alle categorieën exclusief te testen, dat wil zeggen, onafhankelijk van elkaar. In het geval van COVER bijvoorbeeld, zien we dat circulaire regels, circulaire regelparen, contradictoire conclusies en contradictoire regelparen alleen kunnen worden gedetecteerd via de overkoepelende algemene test op circulariteit en contradictie, die computationeel aanzienlijk kostbaarder is. Soortgelijke problemen zien we bij de andere tools.

Bij PROLOGA en mogelijk ook bij VERITAS ontbreekt de definitie van de Input Set en de Goal Set. In geen van de artikelen van de ontwerpers van deze tools worden de Input Set en de Goal Set expliciet genoemd. Hoogst waarschijnlijk wordt ieder attribuut in deze systemen beschouwd als een mogelijke input aan het systeem en is de Goal Set gewoonweg niet aanwezig. Dit is in elk geval zo bij PROLOGA. Elke beslistabel kan worden gekozen als startpunt en vanaf daar worden alle afleidingen gemaakt die mogelijk zijn. Het nadeel van het niet van te voren vaststellen van de Input Set is dat veel redundante regels mogelijk niet kunnen worden gedetecteerd. Immers, in de praktijk kan het zijn dat bepaalde regels of tabellen nooit als startpunt worden gebruikt en bovendien nooit bereikbaar zijn via de overige regels en tabellen. Dit is een belangrijke vorm van redundantie.
Het nadeel van het niet van te voren vaststellen van de Goal Set is dat onvolledigheid mogelijk niet wordt gedetecteerd. De Goal Set geeft namelijk aan welke attributen de mogelijke output van het systeem vormen. Het kan zijn dat een systeem in bepaalde environments geen enkel element uit de Goal Set kan afleiden. Als de Goal Set überhaupt niet is gedefinieerd, dan kan deze vorm van onvolledigheid, onbruikbare output, niet worden gedetecteerd. Wat zeer waarschijnlijk voor alle hier besproken tools geldt, is dat er geen set van gewenste goals kan worden aangegeven voor bepaalde environments. De detectie van ontbrekende output is daardoor in ieder geval onmogelijk.

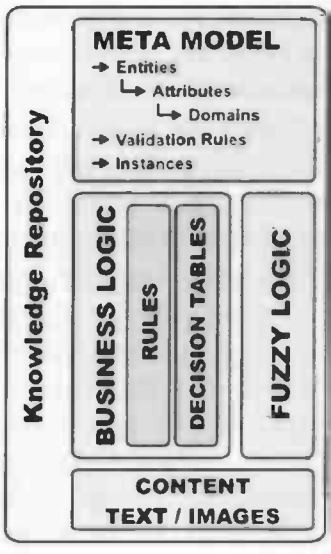
Al het voorgaande in overweging nemende kan worden gesteld dat geen van de onderzochte tools werkelijk geschikt is voor een gedegen anomaliedetectie in de omgeving van het Knowledge Framework van Everest. Zowel de volledigheid van de detectie met betrekking tot alle mogelijke anomaliecategorieën als de complexiteit van de invoer laten te wensen over.

# H4 ONDERZOEK NAAR HAALBAARHEID EN WENSELIJKHEID

In dit hoofdstuk zal eerst dieper worden ingegaan op de werking van het Knowledge Framework van Everest. Vervolgens zal worden aangegeven in hoeverre de verschillende typen van anomalieën zoals die zijn besproken in de theorie van hoofdstuk 2 aan de orde zijn binnen het Knowledge Framework. Bovendien zal worden besproken in hoeverre detectie van de verschillende typen wenselijk is in de praktijk. Als laatste zal worden aangegeven in hoeverre een verificatietool voor alle aspecten van het Knowledge Framework haalbaar is met betrekking tot dit afstudeeronderzoek en de tijd die daar voor staat. Dit hoofdstuk dient als voorbode van het volgende hoofdstuk, waarin op basis van de bevindingen in dit hoofdstuk, zal worden aangegeven aan welke eisen de algoritmen en het prototype van de verificatietool zullen moeten voldoen.

## 4.1 Beschrijving van het Knowledge Framework.

Het Knowledge Framework is te onderscheiden in een aantal onderdelen. Het hart van het systeem wordt gevormd door de Knowledge Repository. De Knowledge Repository bevat business logica en fuzzy logica. De business logica kan vervolgens weer in de vorm zijn van beslistabellen en/of in de vorm van productieregels. Zie ook het schema van Figuur 8. De inference engine beschouwt al deze bronnen in het vormen van een oordeel in een backward chaining modus.




Figuur 8: De Knowledge Repository

### 4.1.1 Het metamodel

In de theorie wordt een KB onderverdeeld in een set van regels en/of beslistabellen en een declaratieset. De declaratieset wordt gevormd door de input set, de goal set en de set van semantische contradicties. In het Knowledge Framework liggen deze zaken iets anders. Er is geen sprake van een declaratieset maar wel van een zogenaamd metamodel. Het metamodel bestaat uit een verzameling declaraties en expressies die als beginconditie gelden. Het metamodel is opgebouwd uit een verzameling entiteiten, attributen, domeinen, validatieregels en instanties. Een entiteit beschikt over een verzameling attributen. Zo kan de entiteit 'wijn' bijvoorbeeld beschikken over de attributen 'wijn.smaak', 'wijn.kleur', 'wijn.prijs' en 'wijn.streek'. Attributen zijn van een bepaald domeintype. Zo heeft het attribuut wijn.kleur bijvoorbeeld het domeintype 'wijnkleuren' en 'wijn.prijs' bijvoorbeeld het domeintype 'currency'. Domeintypen kunnen standaard zijn, zoals integer of currency, of gespecificeerd door de gebruiker in het onderdeel domeinen. Daarin kan aangegeven worden dat er een speciaal domeintype voor bijvoorbeeld 'wijnkleuren' aangemaakt moet worden. Tevens worden hier de mogelijke domeinwaarden

gespecificeerd. In dit geval bijvoorbeeld: 'rood', 'wit' en 'rosé'. De set van attributen met hun bijbehorende domeinwaarden kunnen gezien worden als de input set.

De set van semantische contradicties (constraints) wordt op twee plaatsen aangegeven in het metamodel. Ten eerste is er de mogelijkheid een attribuut al dan niet op 'multivalued' te zetten. Dit wil zeggen dat wanneer bijvoorbeeld het attribuut wijn.kleur niet multivalued is, dat wijn.kleur niet tegelijk 'rood' en 'wit' kan zijn. Ten tweede kunnen er meer complexe contradicties worden aangegeven in het onderdeel validatieregels. Hierin kan bijvoorbeeld worden aangegeven dat wanneer wijn.kleur = 'rood' dat dan voor wijn.smaak alleen nog de waarden 'vol', 'krachtig' en 'soepel' geldig zijn. Feitelijk kunnen in de validatieregels zowel semantische contradicties worden aangegeven (zaken die nooit waar mogen zijn) als semantische tautologieën (zaken die juist altijd waar moeten zijn).

De goal set (de set waarin wordt aangegeven wat de mogelijke einddoelen zijn) is niet aanwezig in het Knowledge Framework. Dit heeft te maken met het feit dat alleen backward chaining wordt ondersteund. Alle attributen kunnen in principe een goal zijn.

### 4.1.2 Business rules

Productieregels worden in het Knowledge Framework *business rules* genoemd. Business rules kunnen in het Knowledge Framework een zeer uitgebreide en complexe set van condities bevatten. De expressie aan de linkerkant van de regel kan zowel logische als wiskundige operatoren bevatten. In feite worden de expressies gedefinieerd in het propositionele deel van de predikatenlogica. De literals hebben structuur en kunnen complexe termen met functiesymbolen bevatten. In de conclusie wordt echter altijd maar één attribuut op een andere waarde gezet. De vorm van de regels in het Knowledge Framework lijkt hiermee dus op de Horn Clause vorm, maar dit is niet noodzakelijk het geval. Aan de linkerkant van de regel is immers elke combinatie van conjuncties of disjuncties mogelijk terwijl de Horn Clause vorm altijd een conjunctie van literals heeft aan de linkerkant van de regel $((p \wedge q \wedge r) \rightarrow s)$ [17].

### 4.1.3 Beslistabellen

Beslistabellen zijn in het Knowledge Framework vrijwel op dezelfde manier opgebouwd als in de definities uit hoofdstuk 2, paragraaf 2.1.2. Zowel limited als extended beslistabellen zijn mogelijk (zowel binaire als meerwaardige condities mogelijk). Verder zijn consolidated beslistabellen mogelijk (in de condities zijn * en [] mogelijk) en zijn beslistabellen in het Knowledge Framework altijd mutually exclusive (elke conditieconfiguratie kent één en slechts één actieconfiguratie). Een verschil met de theorie van beslistabellen is echter de mogelijkheid om complexe expressies te gebruiken in de conditieonderwerpen alsmede in de conditiewaarden.

Een belangrijk punt om op te merken is dat in het Knowledge Framework beslistabellen onderhuids worden gerepresenteerd door een ouder-kinderen-relatie van de verschillende conditie- en actiealternatieven. Het voorbeeld van Figuur 9 hieronder geeft aan wat hiermee wordt bedoeld. Elk conditiealternatief kent een aantal componenten: Een waarde (de expressie), een verwijzing naar een conditieonderwerp

van de tabel en een set van andere conditiealternatieven. Dit laatste component geeft aan wat de kinderen zijn waarvan dit conditiealternatief een ouder is

| Conditieonderwerp 1 | CA(1,1) = A | | CA(1,2) = B | | |
|---|---|---|---|---|---|
| Conditieonderwerp 2 | CA(2,1) = P | CA(2,2) = [ ] | CA(2,3) = P | CA(2,4) = Q | CA(2,5) = R |
| Actieonderwerp 1 | AA(1,1) = U | AA(1,2) = V | AA(1,3) = W | AA(1,4) = X | AA(1,5) = Y |
| *Kolom* | *1* | *2* | *3* | *4* | *5* |

**Conditiealternatief CA(1,1):**

| Waarde | A |
|---|---|
| Conditieonderwerp | Conditieonderwerp 1 |
| Kinderen | { CA(2,1) ; CA(2,2) } |

**Figuur 9: Representatie ouder-kinderen-structuur**

Deze representatie heeft als voordeel dat het testen van volledigheid van beslistabellen hierdoor eenvoudig is te realiseren. Er hoeft dan slechts te worden getest of alle kinderen van een bepaalde ouder het volledige domein van het desbetreffende conditieonderwerp omvatten. Het nadeel van deze representatie is echter dat de kolomstructuur verloren gaat. Deze is natuurlijk wel weer te reproduceren door van alle kinderen vast te stellen wat de ouder is, maar het blijkt niet direct meer uit deze vorm van representatie. In paragraaf 4.3 zal deze problematiek verder worden behandeld met betrekking tot anomaliedetectie.

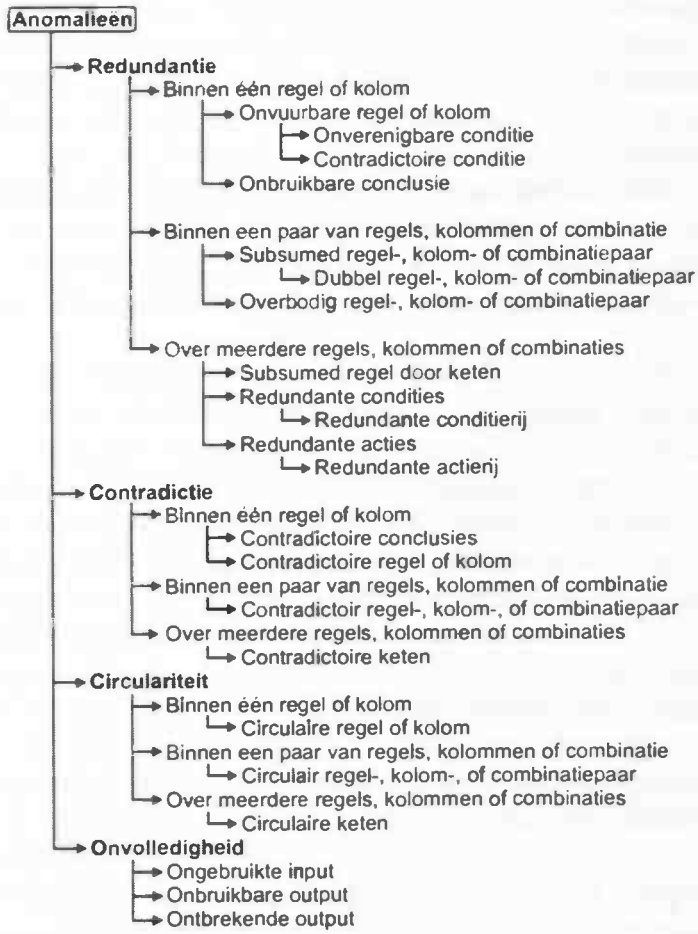
## 4.2 Praktijkervaringen en wenselijkheid van detectie

In hoofdstuk 2 is een theoretisch overzicht gegeven van alle anomalieën die mogelijk zijn in een kennissysteem gebaseerd op regels, beslistabellen of een combinatie van deze twee. De volgende stap is het onderzoeken hoe het in de praktijk gesteld is met deze anomalieën. Het kan bijvoorbeeld zijn dat bepaalde (triviale) anomalieën in de praktijk onmogelijk te creëren zijn vanwege ingebouwde beperkingen in de ontwerpomgeving. Ook kan het zijn dat ontwerpers aan verificatie van bepaalde soorten anomalieën meer waarde hechten dan aan andere vanwege hun schadelijkheid of veelvoorkomendheid in de praktijk. Het uitgangspunt van dit onderzoek is het Knowledge Framework van Everest.

Zes ervaren kennistechnologen van Everest is gevraagd om de beschreven anomalieën uit hoofdstuk 2 door te nemen met betrekking tot het Knowledge Framework. Zij hebben vervolgens hun ervaringen en aanbevelingen gegeven over zaken zoals:

- mogelijkheid van voorkomen
- frequentie van voorkomen
- schadelijkheid
- wenselijkheid van verificatie

Per categorie zoals beschreven in hoofdstuk 2, zullen deze zaken hieronder worden beschreven. We zullen een samenvatting geven van de ervaringen en aanbevelingen van de verschillende kennistechnologen die hebben meegewerkt aan het onderzoek. Figuur 4 op bladzijde 13 wordt hier nogmaals getoond ter herinnering van de verschillende categorieën van anomalieën.




**Figuur 4 (herhaling): Categorisering van anomalieën**

**Redundantie: onverenigbare condities:** De ontwerpomgeving van het Knowledge Framework laat geen ongeldige attributen toe. Tevens kunnen domeinwaarden die niet tot het geldige bereik behoren niet worden ingevoerd. De meest eenvoudige vorm van onverenigbare condities wordt hierdoor reeds voorkomen. Wel is het nog mogelijk om in expressies een onzinnige operator te gebruiken waardoor de gehele conditie alsnog onverenigbaar wordt. Ook kan een conditie onverenigbaar zijn doordat deze nergens in de business logica als conclusie voorkomt. De ervaring is dat deze anomalie niet vaak voorkomt. Echter, detectie is toch gewenst aangezien de schade van een onverenigbare conditie redelijk groot kan zijn omdat de regel onvuurbaar wordt.

**Redundantie: contradictoire condities:** Een semantische of syntactische contradictie in de conditie van een regel is niet zeer waarschijnlijk maar kan wel gecreëerd worden zowel bij regels als tabellen. Vooral complexe semantische contradicties zijn lastig terug te vinden. Verificatie op dit onderdeel is daarom gewenst, mede omdat ook hier de regel onvuurbaar zal zijn en daarom wel degelijk schadelijk kan zijn. Er moet echter opgemerkt worden dat in tabellen syntactische contradicties in de condities alleen te bereiken zijn door het construeren van twee gelijke conditieonderwerpen (tenminste, wanneer als conditiewaarde geen complexe expressie wordt toegelaten). Dit zou eigenlijk überhaupt onmogelijk moeten zijn om te construeren in de ontwerpomgeving.

**Redundantie: onbruikbare conclusies:** Het Knowledge Framework kent geen goal set. In het Knowledge Framework kan alles in principe een mogelijk doel zijn. Een test op deze vorm van redundantie is daarom in principe onmogelijk. Immers, wanneer een bepaalde conclusie van een regel niet in een andere regel als conditie wordt gebruikt, kan deze conclusie nog steeds wel een mogelijk doel zijn. Ook is er het probleem dat bepaalde conclusies binnen de logica van de regels onbruikbaar kunnen zijn maar wel degelijk op andere plaatsen worden gebruikt, bijvoorbeeld het tonen op een scherm. Verificatie op dit onderdeel is op zich wel gewenst maar dus moeilijk uitvoerbaar. In de volgende paragraaf zal hier meer aandacht aan worden besteed.

**Redundantie: subsumed regel-, kolom- of combinatiepaar:** Deze vorm kan voorkomen in het Knowledge Framework. De kans op een dergelijke redundantie is niet groot wanneer ervaren kennistechnologen het systeem ontwikkelen. Maar wanneer ook minder ervaren mensen ontwikkeling of onderhoud plegen aan het systeem wordt de kans op dergelijke anomalieën groter. Normaal gesproken levert deze anomalie geen problemen op, behalve dat de KB meer regels bevat dan nodig is. Echter, wanneer tijdens onderhoud één van de regels wordt verwijderd of gewijzigd, of wanneer een complete module wordt toegevoegd aan een andere module kunnen wel problemen ontstaan. Verificatie op dit type is daarom gewenst.

**Redundantie: dubbel regel-, kolom- of combinatiepaar:** Zie subsumed regel-, kolom- of combinatiepaar. Dezelfde overweging geldt.

**Redundantie: overbodig regel-, kolom- of combinatiepaar:** Deze vorm zal waarschijnlijk niet vaak voorkomen in de praktijk. Ten eerste moeten er een paar van regels, kolommen of een combinatie zijn met een equivalente conclusie, wat al weinig voorkomt. Ten tweede moet het dan ook nog zo zijn dat de conditie van de éne regel of kolom een logische negatie vormt van de conditie van de andere regel of kolom. De kans op het voorkomen hiervan is gering. Bovendien is er normaal gesproken geen schade: het regel-, kolom- of combinatiepaar blijft vuurbaar. Desalniettemin is een test hiervoor waarschijnlijk eenvoudig te construeren en daarom toch wenselijk.

**Redundantie: subsumed regel door keten:** Een moeilijk te ontdekken vorm van redundantie vanwege de complexiteit. Over het algemeen zal deze vorm niet heel veel voorkomen volgens de geïnterviewde kennistechnologen. Wat betreft schadelijkheid en wenselijkheid van detectie gelden dezelfde overwegingen als bij subsumed regel-, kolom- of conbinatiepaar.

**Redundantie: redundante conditierij:** Een redundante conditierij is wel te maken in het Knowledge Framework maar is zeer onwaarschijnlijk. Een dergelijke situatie is meteen zichtbaar tijdens het ontwikkelen (alleen maar sterretjes in de conditiewaarden). De test hierop is waarschijnlijk eenvoudig te maken, maar eigenlijk zou deze situatie überhaupt onmogelijk te creëren moeten zijn in de ontwerpomgeving.

**Redundantie: redundante actierij:** Ook hier geldt dat deze anomalie zeer waarschijnlijk nooit zal voorkomen. Hoewel het in het Knowledge Framework nog wel kan, zou de ontwerpomgeving eigenlijk nooit twee gelijke actieonderwerpen mogen toelaten.

**Contradictie: contradictoire conclusie:** Een contradictoire conclusie is onwaarschijnlijk maar niet onmogelijk om te construeren in het Knowledge Framework. Aangezien contradicties ongewenst gedrag van het systeem zijn en daarom schadelijk, is een verificatie op deze anomalie wenselijk.

**Contradictie: contradictiore regel:** Hierbij geldt dezelfde overweging als bij contradictiore conclusie.

**Contradictie: contradictoir regel-, kolom- of combinatiepaar:** Een aantal kennistechnologen hadden wel degelijk ervaring met deze situatie. Bovendien verwacht men dat wanneer onervaren mensen ontwikkeling of onderhoud plegen, dit probleem vaker zou kunnen voorkomen. Verificatie is dus gewenst, zeker met betrekking tot de schade van contradicties.

**Contradictie: contradictoire keten:** Ook verificatie van contradictie in een keten van meerdere regels werd gewenst bevonden. Er werd echter opgemerkt dat de kans op deze situatie kleiner zal zijn dan op een contradictoir paar van regels. In de praktijk is het zo dat kennissystemen weliswaar een diepe vertakte regelstructuur kunnen hebben, maar meestal zijn er geen elementen in lager gelegen takken die ook voorkomen in hoger gelegen taken.

**Circulariteit:** Voor de drie niveaus van circulariteit (circulaire regel of kolom, circulair regel-, kolom- of combinatiepaar en circulaire keten) gelden dezelfde overwegingen als voor de drie niveaus van contradictie die hierboven zijn beschreven. Verificatie is voor alledrie de niveaus gewenst.

**Onvolledigheid: onbruikbare output:** Volledigheidtests worden zeker belangrijk bevonden. Immers, een onvolledige KB levert in bepaalde gevallen geen output. De test op onbruikbare output is in feite de meest algemene test voor volledigheid. Er is sprake van onbruikbare output wanneer in een bepaald environment geen enkel goal attribuut wordt afgeleid. Tests hiervoor zijn in het Knowledge Framework niet aanwezig, aangezien er geen Goal Set beschikbaar is. Het is dus onmogelijk om deze test uit te voeren. Deze test is wel degelijk gewenst, als uitbreiding op de test voor ongebruikte input.

**Onvolledigheid: ontbrekende output:** Deze anomalie kan worden gezien als de meer specifieke vorm van de anomalie onbruikbare output. Er is sprake van wanneer in bepaalde environments niet elke gewenste output wordt afgeleid. Een test voor deze anomalie kan op zich handig zijn, maar is in de praktijk nooit volledig haalbaar. De gebruiker zal namelijk moeten aangeven voor ieder mogelijk environment wat de gewenste output is voor dat environment. Dit is praktisch onmogelijk aangezien er zeer veel verschillende environments kunnen zijn. Het is wellicht wel haalbaar om bepaalde specifieke environments te testen op ontbrekende output.

**Onvolledigheid: ongebruikte input:** Ongebruikte input wordt in de huidige versie van het Knowledge Framework reeds getest binnen beslistabellen. Echter de eerste conditieregel wordt niet getest op volledigheid om het splitsen van tabellen toe te staan. Dit wil zeggen dat volledigheid over alle tabellen niet gegarandeerd is. Volledigheid van regels wordt in het geheel niet getest. Verificatie op ongebruikte input met betrekking tot de gehele business logica is daarom gewenst.

## 4.3 Haalbaarheid van implementatie

Op basis van het voorgaande zal nu worden besproken in hoeverre de verschillende tests op anomalieën mogelijk zijn met betrekking tot alle details van de omgeving van het Knowledge Framework. Bovendien zal worden besproken of bepaalde zaken haalbaar zijn met betrekking tot een afstudeeronderzoek en de beperkte tijd die daarvoor beschikbaar is.

### Complexe expressies

De theorie van anomalieën in beslistabellen en regels is voornamelijk op basis van de propositielogica [2, 3, 6, 7, 8, 9]. Dit is niet verbazingwekkend. Het detecteren van anomalieën in propositielogische formules is een stuk eenvoudiger dan in complexere representaties zoals de meer objectgeoriënteerde representatie van het Knowledge Framework [18]. Een verificatietool voor het Knowledge Framework zal rekening moeten houden met alle mogelijke attributen bij een entiteit, het attribuuttype en alle mogelijke domeinwaarden bij een attribuut. Bij attributen met het type integer, double of currency moet bovendien rekening worden gehouden met het geldige bereik en wiskundige operaties op het attribuut. Het volgende voorbeeld, dat zeer goed mogelijk is in het Knowledge Framework, laat zien dat het testen van contradictie of redundantie binnen een paar van twee regels aanzienlijk complex kan zijn.

```
IF wijn.prijs <= standaard.marge * (wijn.inkoopprijs * (wijn.BTWtarief + 100) / 100)
THEN
wijn.verkoopmarge = laag

IF wijn.prijs <=  7,50
THEN
wijn.verkoopmarge = laag
```

Het voorbeeld laat zien dat voordat men überhaupt kan testen op anomalieën, de complexe expressies in de condities wiskundig moeten worden vergeleken op equivalentie. Natuurlijk is dit mogelijk, maar het gaat waarschijnlijk te ver voor dit afstudeeronderzoek om ook dergelijke complexe expressies te vergelijken. Daarnaast is het zo dat in de praktijk maar weinig gebruik wordt gemaakt van dergelijke complexe wiskundige expressies. Een versimpeling van het prototype op dit punt is daarom geoorloofd.

## Representatie beslistabellen via ouder-kinderen-structuur

De representatie van beslistabellen bestaat uit een ouder-kinderen-structuur van conditie- en actiealternatieven. Deze representatie is alleen handig bij het testen op volledigheid binnen beslistabellen. Er hoeft dan slechts te worden getest of alle kinderen van een bepaalde ouder het volledige domein van het desbetreffende conditieonderwerp omvatten. Echter, het testen van alle andere categorieën van anomalieën wordt door deze representatievorm juist moeilijker. De meeste andere categorieën hebben namelijk betrekking op anomalieën die binnen een regel plaatsvinden, binnen een paar van regels of over een keten van meerdere regels*. Een regel kan dan in de vorm van een 'echte' productieregel zijn of in de vorm van een kolom in een beslistabel, dat maakt feitelijk niets uit. Het detecteren van bijvoorbeeld een contradictoir, circulair of subsumed regel- kolom- of combinatiepaar vindt plaats op basis van het vergelijken van alle condities en conclusies van twee regels. Wanneer in beslistabellen een regelstructuur aanwezig is in de vorm van kolommen, dan is deze detectie eenvoudig te doen. Het is dan overigens wel noodzakelijk om contracted tabellen eerst om te zetten in extended tabellen (zoals in de figuren 6, 7 en 8 op pagina 13). In het Knowledge Framework zal het echter telkens noodzakelijk zijn bij het detecteren van vrijwel alle categorieën, behalve onvolledigheid, de ouder-kinderen-structuur om te zetten in losse kolommen. Dit is natuurlijk mogelijk maar zeker niet wenselijk. Het beste zou zijn om de volledigheidstests te doen in de huidige representatievorm van ouders en kinderen en de andere categorieën te testen in een representatievorm van losse kolommen. Hierbij krijgen de kolommen uiteraard wel een nummer of ID dat aangeeft in welke beslistabel de kolom zich bevindt, zodat bij detectie van anomalieën ook kan worden aangegeven in welke beslistabel de kolom met de anomalie zich bevindt. Met betrekking tot dit afstudeerproject zal het waarschijnlijk te veel tijd kosten om alle algoritmen en het prototype te construeren op basis van de ouder-kinderen-structuur zoals die in het Knowledge Framework wordt gebruikt.

## Afwezigheid van een goal set

In de vorige paragraaf is reeds duidelijk geworden dat er in het Knowledge Framework geen goal set (lijst met einddoelen) is en dat het testen op onbruikbare conclusies hiermee in principe onmogelijk wordt. In de praktijk is een bepaalde applicatie natuurlijk wel ontworpen voor een beperkt aantal einddoelen. Het is misschien toch mogelijk om deze set te beschrijven. Bovendien is het wellicht mogelijk dat het Knowledge Framework in de toekomst wel wordt uitgerust met een set van einddoelen (waarin tevens wordt aangegeven welke attributen ook buiten de businesslogica gebruikt worden). Hoewel in het Knowledge Framework de goal set nu niet wordt gedefinieerd, is het misschien mogelijk dit wel (handmatig) in het prototype te doen. De tests op anomalieën die gebruik maken van de goal set worden daarmee mogelijk.

---

* *De detectie van een redundante conditierij en een redundante actierij is in principe vrijwel net zo eenvoudig te realiseren in een ouder-kinderen-structuur van alternatieven als in een kolommenstructuur. Hierbij maakt de representatie niet zo veel uit. Deze anomalieën zijn specifiek voor beslistabellen en zouden eigenlijk überhaupt onmogelijk te realiseren moeten zijn in de ontwerpomgeving. Zie ook de vorige paragraaf.*

## Volledigheidtests en defaultwaarden

Een belangrijk punt dat werd genoemd door een aantal kennistechnologen van Everest, is dat het binnen het Knowledge Framework mogelijk is om attributen een defaultwaarde toe te kennen. Vanwege deze mogelijkheid worden bepaalde regels soms weggelaten, aangezien het betreffende attribuut reeds een defaultwaarde heeft die niet meer geconcludeerd hoeft worden door een regel. Zo zijn onvolledige sets van regels mogelijk zonder dat dit een onvolledig gedrag veroorzaakt van het systeem. Een verificatietool die werkzaam is op alle details van het Knowledge Framework zal dus ook hiermee rekening dienen te houden. De haalbaarheid van dit specifieke aspect binnen dit afstudeerproject is twijfelachtig.

## Business rules niet noodzakelijk in conjunctievorm

Business rules in het Knowledge Framework zijn niet noodzakelijk in conjunctievorm. De linkerkant van de formule kan bestaan uit een willekeurige expressie, al dan niet met wiskundige operatoren. Dit zal het testen op anomalieën aanzienlijk complexer maken dan wanneer de regels wel in een vorm met alleen conjuncties van positieve of negatieve literals in het antecedent en de conclusie zouden zijn. Zie ook het voorbeeld bij datastructuur en complexiteit.

## Expressies mogelijk in beslistabellen

Het feit dat in het Knowledge Framework complexe expressies mogelijk zijn in zowel de conditie- en conclusieonderwerpen als de conditie- en conclusiewaarden, maakt verificatie op anomalieën aanzienlijk complexer. Voor dit afstudeeronderzoek zal het niet haalbaar zijn in de tijd die ervoor staat om expressies binnen beslistabellen mee te nemen. Bovendien geldt hier hetzelfde argument dat al genoemd is bij datastructuur en complexiteit, namelijk dat een versimpeling op dit punt geoorloofd is aangezien complexe expressies in de alternatieven van beslistabellen in de praktijk maar weinig worden gebruikt.

# H5 DOELSTELLINGEN & EVALUATIE-CRITERIA

## 5.1 Doelstellingen

Het vorige hoofdstuk gaf een beschrijving van het Knowledge Framework van Everest. Er is aangegeven wat de wensen zijn wat betreft verificatie en wat de haalbaarheid is van een prototype verificatietool voor alle facetten van de business logica van het Knowledge Framework. Naar aanleiding van deze bevindingen zijn de volgende doelstellingen voor de te ontwikkelen algoritmen en het prototype van de verificatietool geformuleerd:

1. **Stand alone:** Het prototype van de verificatietool zal een stand alone applicatie worden. Integratie in het Knowledge Framework is niet haalbaar in de beperkte tijd. Een stand alone applicatie biedt de mogelijkheid om in eerste instantie gebruik te maken van versimpelde en beperktere syntaxis en representaties. Bovendien vervalt zo het risico te verzanden in details van het Knowledge Framework die niets te maken hebben met dit afstudeerproject.

2. **Input:** De algoritmen en het prototype van de verificatietool moeten worden ontwikkeld voor de verificatie van een KB alleen bestaande uit regels en een declaratieset. Beslistabellen kunnen eventueel later nog worden toegevoegd, maar zullen dan een representatie krijgen van losse kolommen vanwege de argumenten die in het vorige hoofdstuk zijn genoemd. Aangezien een representatie via losse kolommen vrijwel gelijk is aan die van regels, heeft de toevoeging van beslistabellen in eerste instantie geen prioriteit. De regels zullen in een vorm zijn waarbij alleen conjuncties van losse literals zijn toegestaan, zowel in het conditiegedeelte als in het conclusiegedeelte: $Lit_1 \wedge ... \wedge Lit_n \rightarrow Lit_1 \wedge ... \wedge Lit_m$. Het toestaan van meerdere literals in de conclusie zorgt reeds voor een sterke overeenkomst met de representatie van kolommen in een beslistabel. De declaratieset moet bestaan uit een attributenset van geldige attributen en bijbehorende domeinen. In eerste instantie zullen alleen attributen van het type string mogelijk zijn. Bij elk attribuut zal worden aangegeven of deze multi- of singlevalued is en wat de mogelijke domeinwaarden zijn van het attribuut. Later kunnen eventueel attributen van het type integer worden toegevoegd om ook te kunnen werken met domeinen die bestaan uit een waardebereik. Verder zal er een goalset en een inputset worden gedefinieerd die beide een subset zullen zijn van attributenset. Hoewel het Knowledge Framework geen goal- en inputset bevat, zal het prototype hier dus wel gebruik van maken. Tevens zal een set van semantische contradicties worden gedefinieerd.

3. **Representatie van literals:** De input en de algoritmen zullen in eerste instantie op basis van de syntaxis van de propositielogica zijn. De gebruikte literals zullen echter in de volgende, iets complexere vorm zijn: valentie(attribuut relatie waarde). Het literal kan een negatieve of positieve valentie hebben (valentie kan TRUE of FALSE zijn) en bestaat uit een attribuut dat wordt vergeleken met een waarde door een binaire relatie. In eerste instantie zal als relatie alleen de 'is-gelijk-aan' worden gebruikt (=).

Wanneer het prototype correct werkt op basis van deze versimpelde syntaxis, kan altijd nog gekozen worden voor een uitbreiding die meer overeenkomt met alle facetten van Knowledge Framework. Een uitbreiding waarbij ook de relaties (<), (>), (≤) en (≥) mogelijk zijn, zal dan als eerste worden geïmplementeerd.

4. **Anomalieën:** De te ontwikkelen algoritmen en het prototype van de verificatietool moeten de anomalieën kunnen detecteren zoals aangegeven in Tabel 10. In deze tabel staan alle subcategorieën uit Figuur 4 op bladzijde 13. De detectie zal zoals gezegd alleen op basis van regels zijn.

5. **Output:** Het prototype zal bij de detectie van een anomalie een melding moeten geven van welk type anomalie sprake is en welke regels de anomalie veroorzaken.

6. **Testvoorbeelden:** Er zal een set van testregels en declaratiesets moeten worden geconstrueerd om de verificatie van elk type anomalie te kunnen testen. De testvoorbeelden moeten bestaan uit voorbeelden die precies één specifieke anomalie bevatten en uit voorbeelden die vele verschillende categorieën van anomalieën bevatten. Voor het construeren van de testvoorbeelden zal assistentie worden gevraagd van een aantal ervaren kennistechnologen van Everest.

| Type Anomalie | Detectie |
|---|---|
| ***Redundantie*** | |
| Onverenigbare condities | Alleen regels |
| Contradictoire condities | Alleen regels |
| Onbruikbare conclusies | Alleen regels |
| Subsumed regel-, kolom- of combipaar | Alleen regels |
| Dubbel regel-, kolom- of combipaar | Alleen regels |
| Overbodig regel-, kolom- of combipaar | Alleen regels |
| Subsumed regel of kolom door keten | Alleen regels |
| Redundante conditienj | Nee |
| Redundante conclusienj | Nee |
| ***Contradictie*** | |
| Contradictoire conclusies | Alleen regels |
| Contradictoire regels | Alleen regels |
| Contradictoir regel-, kolom- of combipaar | Alleen regels |
| Contradictoire keten | Alleen regels |
| ***Circulariteit*** | |
| Circulaire regel of kolom | Alleen regels |
| Circulair regel-, kolom- of combipaar | Alleen regels |
| Circulaire keten | Alleen regels |
| ***Onvolledigheid*** | |
| Ongebruikte input | Alleen regels |
| Onbruikbare output | Alleen regels |
| Ontbrekende output | Alleen regels |

**Tabel 10: Detectiedoelstellingen**

## 5.2 Evaluatiecriteria van het prototype

1. **Aantal gedetecteerde categorieën:** Het prototype zal ten eerste worden geëvalueerd op het aantal verschillende categorieën van anomalieën dat het kan detecteren. Het criterium is dat alle categorieën die zijn genoemd in de doelstellingen gedetecteerd moeten kunnen worden.

2. **Percentage correcte detectie per categorie:** Ten tweede zal worden geëvalueerd of bepaalde categorieën altijd worden gedetecteerd of dat alleen een bepaald gedeelte van de testvoorbeelden wordt gedetecteerd. Het criterium hierbij is dat

verreweg het grootste deel van de voorbeelden van elk type anomalie gedetecteerd moet kunnen worden.

3. **Percentage foutieve detectie per categorie:** Het kan zijn dat de tool aangeeft dat er een anomalie van een bepaalde categorie aanwezig is, terwijl dit na bestudering in werkelijkheid niet het geval is. Het percentage van dit soort foutieve detectie (false positives) moet nul zijn of in elk geval uiterst laag. Een foutieve melding is namelijk potentieel zeer schadelijk. De gebruiker zou een goede regel kunnen verwijderen of wijzigen naar aanleiding van een foutieve detectie.

4. **Bruikbaarheid:** De verificatietool zal worden geëvalueerd op snelheid en daarmee dus bruikbaarheid. Het testen op de aanwezigheid van anomalieën in een gemiddelde KB mag niet langer duren dan enkele minuten tot een kwartier (op een gemiddelde computer, in de orde van een Pentium 3). De specificaties van een gemiddelde, representatieve KB zoals die worden gebouwd door Everest zijn te zien in Tabel 11*.

| | |
|---|---|
| Totaal aantal attributen: | 200 |
| Waarvan eventueel inputattributen voor de BL: | 50 |
| Inputattributen met waardebereik: | 40 |
| Inputattributen met vaste domeinwaarden: | 10 |
| Gemiddeld aantal domeinwaarden: | 5 |
| Aantal business rules: | 30 |
| Gemiddeld aantal condities van business rule: | 3 |
| Gemiddeld aantal conclusies van business rule: | 2 |
| Aantal beslistabellen: | 100 |
| Gemiddeld aantal conditieonderwerpen: | 3 |
| Gemiddeld aantal actieonderwerpen: | 2 |
| Gemiddeld aantal kolommen | 10 |
| Totaal aantal regels (kolommen x DT's + business rules) | 1030 |
| Gemiddeld aantal condities | 3 |
| Gemiddeld aantal conclusies | 2 |

**Tabel 11: Eigenschappen van een gemiddelde KB**

* *De specificaties van de gemiddelde beslistabel zijn ontstaan vanuit zeer grove schattingen van kennistechnologen van Everest. In de praktijk bestaan er zeer grote verschillen tussen de eigenschappen van KB's. De waarden in deze tabel moeten dus slechts worden geïnterpreteerd als een indicatie van de orde van grootte. Verder kan worden opgemerkt dat het lastig bleek een schatting te maken van welk deel van de attributen daadwerkelijk gebruikt wordt in beslistabellen en business rules. Vaak worden attributen namelijk alleen buiten de Business Logica gebruikt in het Knowledge Framework (bijvoorbeeld in schermen). Het aantal gebruikte attributen in de Business Logica die input kunnen zijn aan het systeem is in ieder geval slechts een fractie van het totaal aantal attributen.*

# H6 MODEL & ALGORITMEN

In het vorige hoofdstuk zijn de doelstellingen geformuleerd waaraan de algoritmen en het prototype van de verificatietool moeten voldoen. Op basis van deze doelstellingen en de theorie uit hoofdstuk 2 konden verschillende concrete algoritmen worden opgesteld voor het detecteren van anomalieën. Dit hoofdstuk geeft een beschrijving van deze algoritmen. In de eerste paragraaf wordt het model beschreven. Het model geeft aan hoe alle verschillende onderdelen, zoals de Rule Base en de declaratieset worden gerepresenteerd. Dit model is vervolgens het uitgangspunt voor de detectiealgoritmen van de verschillende anomaliecategorieën die volgen in de paragrafen daarna.

## 6.1 Model

In het model, dat zoals gezegd de basis vormt voor de algoritmiek, is een scheiding tussen de Rule Base en de declaratieset. De Rule Base bestaat uit regels die de vorm hebben:

$$Lit_1 \wedge Lit_2, \wedge \ldots \wedge Lit_n \rightarrow Lit_1 \wedge Lit_2, \wedge \ldots \wedge Lit_m$$

Hierbij is $Lit_x$ een literal in de vorm:

valentie(attribuutnaam relatie attribuutwaarde)

In dit model is er voor gekozen om literals te definiëren van twee typen: literals van het type string, bijvoorbeeld (kleur = rood) en literals van het type integer, bijvoorbeeld (prijs > 150). De relaties <, >, >= en <= zijn alleen toepasbaar op literals van het type integer. Uiteraard zijn er meerdere typen denkbaar voor een literal, maar we zullen in het model eerst alleen uitgaan van de typen string en integer.

De declaratieset bestaat uit een Input Set, een Goal Set en een Semantische Contradicties Set. De Input Set en de Goal Set zijn beide een subset van de Attributen Set. In de Attributen Set wordt aangegeven wat alle geldige attributen zijn. Attributen kunnen wederom van het type string of van het type integer zijn. Beide typen worden als volgt gedeclareerd:

*Attribuut type string*: attribuutnaam, {$waarde_1$, ... , $waarde_m$}, multivalued = false
*Attribuut type integer*: attribuutnaam, min = 0, max = 1000

De Semantische Contradicties Set bestaat uit een lijst van conjuncties van literals die een semantische contradictie vormen:

$$Lit_1 \wedge Lit_2, \wedge \ldots \wedge Lit_x$$

In Figuur 10 wordt het model gegeven van de Rule Base, de declaratieset en al hun onderdelen zoals die hierboven zijn besproken. Alle verschillende componenten worden aangegeven met voorbeeldwaarden.

**Rule Base**

| | |
|---|---|
| Regellijst | $\{R_1, R_2, ..., R_{10}\}$ |

**Regel**

| | |
|---|---|
| ID | 004 |
| Condities | $\{Lit_a, Lit_b, Lit_c\}$ |
| Conclusie | $\{Lit_p, Lit_q\}$ |

**Literal**

| | |
|---|---|
| Valentie | TRUE |
| Attribuut | "wijnkleur" |
| Relatie | "=" |
| Attribuutwaarde | "rood" |

**Declaratieset**

| | |
|---|---|
| Attributen Set | $\{Attr_1, ..., Attr_{10}\}$ |
| Input Set | $\{Attr_3, Attr_4\}$ |
| Goal Set | $\{Attr_7, Attr_8, Attr_{10}\}$ |
| Sem. Contradictie Set | $\{ \{Lit_s, Lit_t\}, \{Lit_x, Lit_y\} \}$ |

**Attribuut_String**

| | |
|---|---|
| AttribuutNaam | "wijnkleur" |
| Attribuutwaarden | {"rood", "wit", "rose"} |
| Multivalued | FALSE |

**Attribuut_Integer**

| | |
|---|---|
| AttribuutNaam | "prijs" |
| Minimumwaarde | 0 |
| Maximumwaarde | 1000 |

**Figuur 10: Model**

## 6.2 Algemeen inzetbare algoritmen

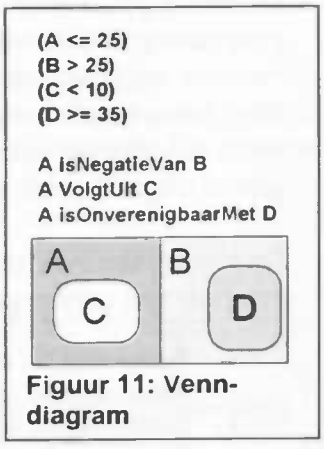
Voordat de algoritmen voor de daadwerkelijke detectie van anomalieën zullen worden behandeld, worden in deze paragraaf eerst een aantal algemeen inzetbare algoritmen beschreven. Deze algoritmen detecteren zelf geen anomalieën, maar kunnen gebruikt worden binnen de algoritmen die wel anomalieën detecteren. In het prototype zullen deze algoritmen vertaald worden in algemeen inzetbare functies.
Ten behoeve van de leesbaarheid is ervoor gekozen om deze algoritmen niet volledig uit te schrijven, maar om alleen voorbeelden te geven van de werking. Een volledige beschrijving in pseudo-code van alle algoritmen is opgenomen in Appendix C. De volgende algemeen inzetbare algoritmen zijn opgesteld:

- **IsGelijkAan (Literal A, Literal B)**
  Vb: IsGelijkAan ( (type = 'groot'), (type = 'groot') )
  IsGelijkAan ( (prijs >= 25), ¬(prijs < 25) )

- **IsNegatieVan (Literal A, Literal B)**
  Vb: IsNegatieVan ( (type = 'klein'), ¬(type = 'klein') )
  IsNegatieVan ( ¬(prijs = 10), (prijs = 10) )
  IsNegatieVan ( (prijs >= 10), (prijs < 10) )

- **VolgtUit (Literal A, Literal B)** *Literal A volgt uit Literal B*
  Vb: Stel: kleur is een singlevalued attribuut.
  VolgtUit (¬(kleur = 'geel'), (kleur = 'blauw') )
  VolgtUit ( (prijs <= 25), (prijs < 20) )
  VolgtUit ( (prijs < 25), ¬(prijs > 20) )

- **IsOnverenigbaarMet (Literal A, Literal B)** *Literal A is onverenigbaar met Literal B*
  Vb: IsOnverenigbaarMet ( (prijs = 10), (prijs > 20) )
  IsOnverenigbaarMet ( (prijs < 10), ¬(prijs <= 10) )



**Figuur 11: Venn-diagram**

In Figuur 11 worden enkele voorgaande algoritmen (negatie, volg uit en onverenigbaar) nader toegelicht middels een Venn-diagram.

- **IsElementInputSet (Literal)** *Literal behoort tot de set van mogelijke input*
  Vb: Input Attribuut: kleur {rood, geel, groen, blauw}
  IsElementInputSet ( (kleur = 'rood') )

- **IsElementGoalSet (Literal)** *Literal behoort tot de set van mogelijke output*

- **BevatSyntactischeContradictie (Literalset)** *De set bevat een syntactische contradictie*
  Vb: BevatSyntactischeContradictie ( (kleur = 'rood'), ¬(kleur = 'rood'), (prijs = 10) )
  BevatSyntactischeContradictie ( ¬(prijs > 10), (type = 'klein'), (prijs = 15) )

- **BevatSemantischeContradictie (Literalset)** *De set bevat een semantische contradictie*
  Vb: Stel: kleur is een singlevalued attribuut.
  BevatSemantischeContradictie ( (kleur = 'rood'), (prijs = 5), (kleur = 'groen') )
  Vb: Stel: semantische contradictie: (kleur = 'geel') ∧ (prijs < 10)
  BevatSemantischeContradictie ( (kleur = 'geel'), (prijs = 5), (type = 'middel') )

- **BevatContradictie (Literalset)** *De set bevat een syntactische of semantische contradictie*

- **IsSubsetVan (Literalset A), (Literalset B)** *Set A is een subset van set B*
  Vb: IsSubsetVan( (C, A), (A, B, C, D) )
  Vb: IsSubsetVan( (A, B, C), (C, A, B) )

## 6.3 Algoritmen voor integriteittests en regeltests

De algoritmen voor integriteittests detecteren anomalieën die zich binnen één regel manifesteren. Het gaat hierbij om de detectie van de volgende anomalieën:

- **Redundantie: onverenigbare condities**
- **Redundantie: contradictoire condities**
- **Redundantie: onbruikbare conclusies**
- **Contradictie: contradictoire conclusies**
- **Contradictie: contradictoire regel**
- **Circulariteit: circulaire regel**

Deze tests kenmerken zich door een vergelijking van alleen de antecedenten en/of de conclusies binnen één regel. Vervolgens worden deze tests voor alle regels in de Rule Base uitgevoerd. De algemeen inzetbare algoritmen kunnen hierbij van dienst zijn. Zo kunnen bijvoorbeeld contradictoire condities en contradictoire conclusies worden gedetecteerd door eenvoudigweg respectievelijk alle antecedenten of alle conclusies van een regel als input mee te geven aan **BevatContradictie(Literalset)**. En zo zijn de algoritmen **IsElementInputSet(Literal)** en **IsElementGoalSet(Literal)** bijvoorbeeld in te zetten voor de detectie van onverenigbare condities en onbruikbare conclusies. Voor de geheel uitgeschreven algoritmen wordt verwezen naar Appendix C.

De algoritmen voor regeltests detecteren alle anomalieën die zich manifesteren binnen een paar van twee regels. Het gaat dan om de detectie van de volgende anomalieën:

- **Redundantie: subsumed of dubbel regelpaar**
- **Redundantie: overbodig regelpaar**
- **Contradictie: contradictoir regelpaar**
- **Circulariteit: circulair regelpaar**

De algoritmen kenmerken zich ditmaal door een vergelijking van een regel met alle andere regels uit de Rule Base. Deze vergelijking wordt vervolgens voor alle regels in de Rule Base uitgevoerd. De algemeen inzetbare algoritmiek wordt hierbij wederom gebruikt. Een subsumed regelpaar kan bijvoorbeeld worden gedetecteerd door vast te stellen of **IsSubsetVan (antecedenten regel 1), (antecedenten regel 2)** en

**IsSubsetVan (conlusies regel 2), (conclusies regel 1)** beide geldig zijn. In dat geval wordt regel 2 subsumed door regel 1. Wanneer tevens het aantal antecedenten en het aantal conclusies van beide regels gelijk is, moet er sprake zijn van een dubbel regelpaar.
Het algoritme **BevatContradictie(Literalset)** wordt uiteraard ingezet in het algoritme voor de detectie van een contradictoir regelpaar. Voor de precieze werking van de algoritmen voor alle regeltests wordt wederom verwezen naar Appendix C.

## 6.4 Algoritmen voor regelextensietests

De detectie van anomalieën in een keten van regels (regelextensietests) vereist een geheel andere aanpak dan de detectie van anomalieën binnen een regel of een regelpaar. Bij de laatste is het voldoende om regels te vergelijken met de declaratieset en/of regels te vergelijken met elkaar. De detectie van anomalieën in regelketens is complexer. Er moet dan worden vastgesteld welke regels gevuurd kunnen worden bij elk mogelijk *environment*. Een environment is een combinatie van inputliterals die samen geen syntactische of semantische contradictie vormen. (zie pagina 7, definitie 4). De sequentie van regels die gevuurd kan worden op basis van een environment wordt de *regelextensie* genoemd. De algoritmiek voor regelextensietests kenmerkt zich door het nemen van drie stappen:

1. Creatie van alle mogelijke environments
2. Vaststellen van de regelextensies voor al deze mogelijke environments
3. Detectie van anomalieën op basis van de regelextensies

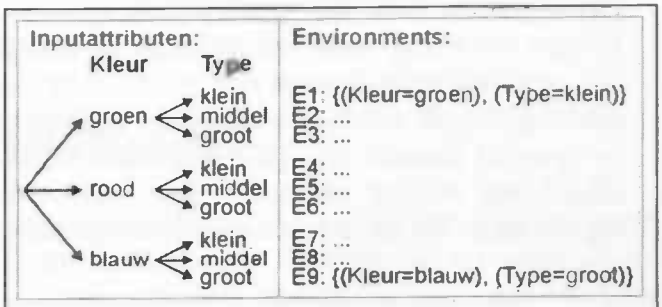
De volgende anomalieën kunnen op deze wijze worden gedetecteerd:

- **Redundantie: subsumed regel door keten**
- **Contradictie: contradictoire keten**
- **Circulariteit: circulaire keten**

De drie stappen van het detectieproces zullen hieronder nader worden uitgewerkt. Wederom ten behoeve van de leesbaarheid wordt hier alleen beschreven hoe de algoritmiek globaal in zijn werk gaat, en wordt de lezer naar Appendix C verwezen voor de precieze uitwerkingen van de algoritmen.

### 6.4.1 Creatie van alle mogelijke environments

Om alle environments te creëren, wordt per inputattribuut bekeken welke mogelijke waarden dit attribuut kan hebben. Bij attributen met vaste domeinwaarden (van het type string) worden de environments opgebouwd volgens een soort boomstructuur. Elk inputattribuut zorgt voor een nieuw stel takken in de boom, want voor elke mogelijke domeinwaarde van het attribuut wordt een nieuw environment gecreëerd. Wanneer er bijvoorbeeld twee inputattributen zijn met



**Figuur 12: Opbouwen environments**

elk drie domeinwaarden, dan zijn er negen mogelijke environments (zie Figuur 12). Het is duidelijk dat het aantal environments exponentieel stijgt met het aantal inputattributen. Een Input Set met 10 (singlevalued) inputattributen met gemiddeld 3 domeinwaarden leidt al tot $3^{10} \approx 59.000$ mogelijke environments! We zullen later nog terugkomen op deze problematiek.
Voor attributen met een waardebereik (van het type integer) ligt de creatie van environments iets anders. Het aantal mogelijke waarden van het inputliteral is dan zeer groot en zelfs oneindig als er geen minimum- en/of maximumwaarde is gedefinieerd. Er kunnen echter wel relevante 'testenvironments' worden samengesteld op basis van de antecedenten van alle regels in de Rule Base. Stel dat er de volgende twee inputattributen zijn van het type integer: *prijs* en *budget*. En stel dat alleen de volgende regels uit de Rule Base antecedenten hebben die refereren aan deze attributen:

ALS (prijs = 85) DAN ...
ALS NIET(prijs = 85) DAN ...
ALS (budget >= 100) DAN ...
ALS (budget < 100) DAN ...

Dan zijn de volgende environments volledig met betrekking tot de verschillende output van het systeem:

environment 1: (prijs = 85), (budget = 99)
environment 2: (prijs = 85), (budget = 100)
environment 3: (prijs = 86), (budget = 99)
environment 4: (prijs = 86), (budget = 100)

De methode werkt dus als volgt: voor elk relevant antecedentliteral wordt een environment gecreëerd dat voldoet aan de relatie van het antecedentliteral en één die juist *niet* voldoet aan die relatie.

Bij de creatie van environments zijn in de praktijk nog een aantal andere aspecten van belang. Wat te doen bijvoorbeeld met attributen die meerdere waarden tegelijk kunnen hebben (multivalued attributen)? Een andere kwestie is dat er geen contradicties mogen voorkomen in de environments. Oplossingen voor deze zaken zullen aan bod komen in het volgende hoofdstuk.

### 6.4.2 Vaststellen van de regelextensies

Het algoritme voor een regelextensie creëert een lijst van alle regels die gevuurd kunnen worden op basis van een bepaald environment. Een regel is vuurbaar wanneer alle conditieliterals element zijn van de Input Set en/of de set van conclusies van reeds eerder gevuurde regels. Dit algoritme is *forward chaining* (gedreven vanuit de input) en opereert *breadth first* (alle regels die vuurbaar zijn op een bepaald niveau in de afleidketen worden gevuurd). Een voorbeeld van het tot stand komen van een regelextensie via de zojuist beschreven methode is te zien in Figuur 13.

| Regel-nummer: | Rule Base: | Environment: |
|---|---|---|
| 1 | A → B | {A, F} |
| 2 | D → E | |
| 3 | B → C | |
| 4 | E & F → ¬C | |
| 5 | C → D | |

| Stap: | Regel: | Afgeleide conclusies: | Gevuurde regels: |
|---|---|---|---|
| 1 | 1 | {B} | {1} |
| 2 | 3 | {B, C} | {1, 3} |
| 3 | 5 | {B, C, D} | {1, 3, 5} |
| 4 | 2 | {B, C, D, E} | {1, 3, 5, 2} |
| 5 | 4 | {B, C, D, E, ¬C} | {1, 3, 5, 2, 4} |
| 6 | - | {B, C, D, E, ¬C} | {1, 3, 5, 2, 4} |

**Figuur 13: Regelextensie**

### 6.4.3 Detectie van anomalieën op basis van de regelextensies

In Figuur 13 is tevens te zien hoe een regelextensie kan worden gebruikt om een contradictoire keten te detecteren. Wanneer de afgeleide conclusies van de gevuurde regels een contradictie vormen, dan is er een contradictoire keten gedetecteerd. Deze keten begint bij de regel die het eerste literal van de contradictie bevat en eindigt bij de regel die het laatste literal van de contradictie bevat. In het voorbeeld van Figuur 13 gaat het om de regelketen: 3, 5, 2, 4. De detectie van een circulaire keten gaat vrijwel overeenkomstig. De compleet uitgewerkte algoritmen in pseudo-code voor de detectie van contradictoire en circulaire ketens staan in Appendix C.

Het algoritme voor de detectie van een subsumed regel door een keten van andere regels onderscheidt zich op een belangrijk punt van de algoritmen voor circulaire keten en contradictoire keten. Een circulariteit of een contradictie mag nooit voorkomen. Detectie van een circulaire keten of contradictoire keten in slechts *één* van de environments is daarom voldoende reden om een melding te geven. Een redundante regel moet echter in *alle* environments redundant zijn. Het algoritme dat is opgesteld voor de test op subsumed regels werkt daarom als volgt: er wordt getest voor alle regels of het verwijderen van de desbetreffende regel uit de Rule Base een andere verzameling afgeleide conclusies veroorzaakt in de regelextensies van alle environments. Er wordt dus gebruik gemaakt van de algemene definitie voor redundantie (hoofdstuk 2, paragraaf 2.2.2, definitie 1). Voordat deze test wordt uitgevoerd, wordt echter eerst getest of de desbetreffende regel geen onverenigbare of contradictoire condities bevat, geen onbruikbare conclusies bevat, niet subsumed is door een andere regel en niet behoord tot een overbodig regelpaar. Zo wordt overlap met de andere redundantietests voorkomen. Wanneer er geen sprake is van dergelijke overlap, maar de afgeleide conclusies van de regelextensies van alle mogelijke environments zijn gelijk, met of zonder de aanwezigheid van de regel, dan is de regel redundant. Een keten van andere regels kan dan in elk environment tot dezelfde conclusies komen.

## 6.5 Algoritmen voor onvolledigheidtests

Het testen van onvolledigheid is in een bepaalde zin onvergelijkbaar met het testen van de andere drie hoofdcategorieën, redundantie, contradictie en circulariteit. Redundantie, contradictie en circulariteit manifesteren zich in een regel, een regelpaar of in meerdere regels. Onvolledigheid heeft te maken met de *afwezigheid* van regels die er eigenlijk wel zouden moeten zijn. De categorieën die gedetecteerd moeten worden zijn:

- **Ongebruikte input**
- **Onbruikbare output**
- **Ontbrekende output**

Het algoritme voor ongebruikte input test of elk mogelijk inputliteral behoort tot de Goal Set of gebruikt wordt in één van de regels als conditie. Als niet aan één van deze voorwaarden voldaan wordt, dan is er sprake van ongebruikte input.

Het algoritme voor onbruikbare output, test of via alle mogelijke environments minimaal *één* van de goal attributen wordt afgeleid. Ontbrekende output gaat nog een stap verder en test of *alle* gewenste goalattributen behorend bij een bepaald environment worden afgeleid. Beide algoritmen werken als volgt: voor alle mogelijke environments worden regelextensies opgebouwd. Vervolgens testen de algoritmen of de betreffende goalattributen aanwezig zijn in de verzameling van de conclusies van deze regelextensies. Deze algoritmen maken dus gebruik van de creatie van environments en regelextensies op basis van die environments. Daarmee lijken qua structuur erg op de regelextensietests. De lezer wordt verwezen naar Appendix C voor de precieze uitwerking van de algoritmen voor onvolledigheidtests.

# H7 IMPLEMENTATIE

## 7.1 Keuze voor Java

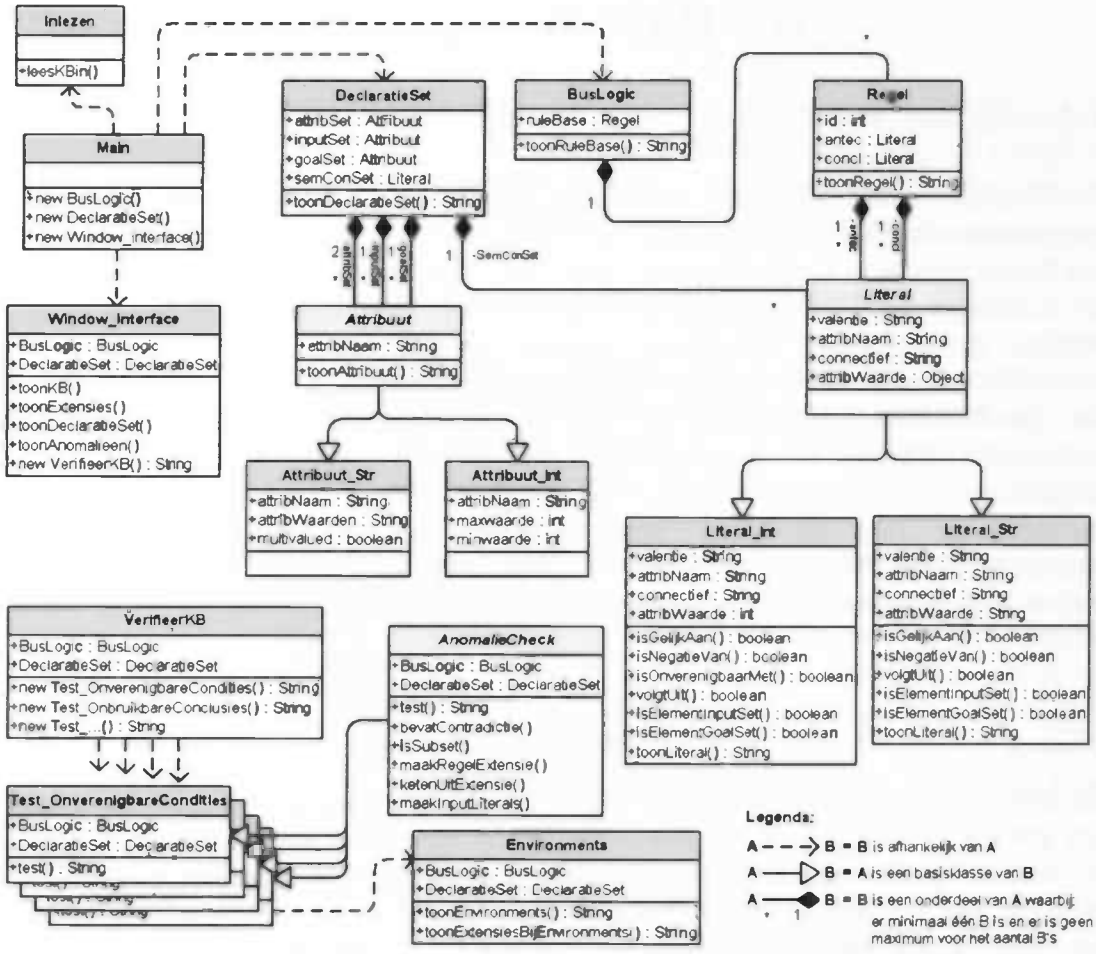
De implementatie van het prototype is gedaan in de objectgeoriënteerde programmeertaal Java. De objectgeoriënteerde structuur van Java is zeer geschikt om een basisstructuur te creëren van alle benodigde onderdelen. Zo kunnen de Rule Base, de declaratieset en de verschillende anomalietests overzichtelijk in aparte klassen worden geconstrueerd. Daarnaast kunnen gemakkelijk de relaties tussen de verschillende onderdelen worden aangegeven en kunnen de klassen uitgerust worden met geschikte methoden, speciaal voor die klasse. Java in combinatie met het programma JBuilder (er is gebruik gemaakt van JBuider 8) levert een krachtige programmeeromgeving. Bovendien is het Knowledge Framework van Everest eveneens in Java gebouwd. Implementatie van het prototype in dezelfde programmeertaal als het Knowledge Framework vergemakkelijkt een eventuele toekomstige opname van de tool of gedeelten daarvan in het Knowledge Framework.

## 7.2 Klassenstructuur

De eerste en tevens zeer belangrijke stap in het implementatieproces is het kiezen van een geschikte klassenstructuur. Het model en de algoritmen uit het vorige hoofdstuk vormen de basis voor de klassenstructuur die uiteindelijk gekozen is. Waar dit mogelijk was vormen de verschillende onderdelen van de representatie uit het vorige hoofdstuk aparte klassen in het prototype. Het UML-schema uit Figuur 14 op de volgende pagina geeft aan uit welke klassen het programma bestaat, welke onderlinge relaties er zijn en welke methoden de klassen bevatten. Klasse-attributen worden in het bovenste gedeelte van de klassen aangegeven, methodes in het onderste gedeelte.

### 7.2.1 Model

De klassen die als eerste zijn geïmplementeerd zijn de klassen die de KB representeren. Deze wordt gevormd door de klassen BusLogic (Business Logic) en DeclaratieSet. BusLogic bestaat uit een lijst van regels; de Rule Base. Regels worden gerepresenteerd door de klasse Regel. Deze bestaat uit twee lijsten van literals, de antecedenten en de conclusies, en een id-nummer. De klasse Literal is een zogenaamde basisklasse (*base class*) die boven de klassen Literal_Str (Literals van het type string) en Literal_Int (Literals van het type integer) staat. De klasse Literal bevat een attribuutnaam, een attribuutwaarde, een relatie en een validatie (precies zoals beschreven in het model uit paragraaf 6.1). De klasse DeclaratieSet wordt gevormd door drie lijsten van attributen: de Attributen Set, de Input Set en de Goal Set, en een lijst van semantische contradicties. Een semantische contradictie bestaat vervolgens weer uit een lijst van literals. De klasse Attribuut is (analoog aan Literal) een basisklasse die boven de klassen Attribuut_Str en Attribuut_Int staat. Attribuut_Str bevat een attribuutnaam, een lijst van geldige attribuutwaarden (het domein) en een booleanvariabele die aangeeft of het attribuut al dan niet multivalued is. Attribuut_Int bevat alleen de attribuutnaam, een maximale waarde en een minimale waarde.




**Figuur 14: UML-diagram van de klassenstructuur van het prototype**

## 7.2.2 Anomalietests

De representatie van de KB is vervolgens het uitgangspunt voor de implementatie van de klassen en methoden die de anomaliedetectie verzorgen. Er is gekozen voor een klassenstructuur waarbij elk type anomalie door een aparte klasse getest wordt. Het voordeel hiervan is dat elke test gemakkelijk afzonderlijk aangeroepen kan worden, afhankelijk van de voorkeur van de gebruiker. Het betekent ook dat het testen van de verschillende anomalieën altijd serieel plaatsvindt. Dat wil zeggen: pas als de voorgaande test helemaal is afgerond, kan de volgende beginnen.* Alle testklassen krijgen als input dezelfde Rule Base en declaratieset mee. Ze bevatten slechts één methode: test(). Deze methode test op het voorkomen van de desbetreffende anomalie, gegeven de Rule Base met bijbehorende declaratieset. Als dit het geval is geeft de methode als output een korte uitleg waarin wordt vermeld waar en waarom

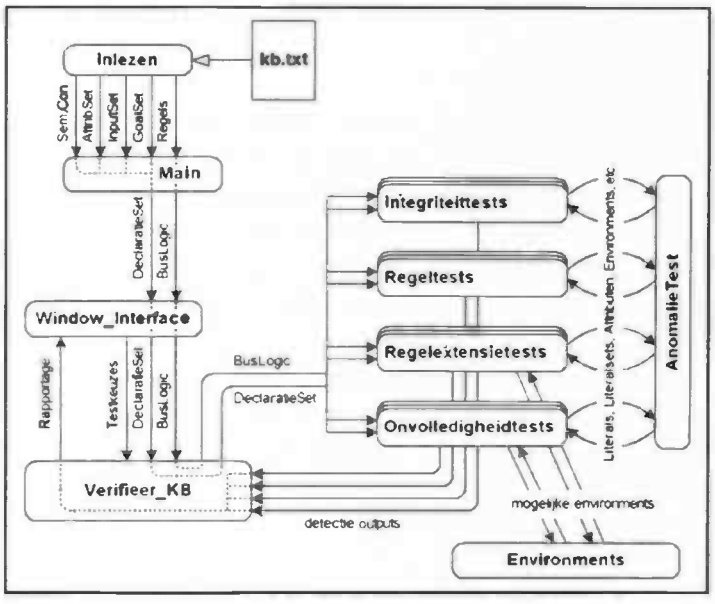
* *Dit houdt in dat bij het testen van alle categorieën van anomalieën telkens opnieuw dezelfde regels in de Rule Base langs worden gelopen (telkens weer opnieuw dezelfde FOR-lus). Theoretisch was een implementatie waarbij alle integriteittest en regeltests plaatsvinden binnen dezelfde (overkoepelende) FOR-lus efficiënter geweest. In de praktijk is de tijdsduur van het aflopen van een FOR-lus echter verwaarloosbaar ten opzichte van de daadwerkelijke anomalietests, en was een losse, gestructureerde implementatie van alle tests te prefereren.*

de anomalie plaatsvond. Wanneer de anomalie vaker voorkomt, op meerdere plaatsen in de Rule Base, wordt van elk voorkomen apart melding gemaakt.

Bij de implementatie van deze methodes bleken de algoritmen van de anomalietests uit het vorige hoofdstuk een goede, directe basis. Aangezien de beschrijvingen van de algoritmen in pseudo-code (zie Appendix C) reeds behoorlijk uitgebreid waren, konden vele vrijwel probleemloos direct in programmacode worden vertaald. De meeste anomalietests in het prototype werken daarom vrijwel precies zo als het bijbehorende algoritme. Alle testklassen zijn bovendien dochters van de basis klasse AnomalieCheck. AnomalieCheck bevat een aantal algemeen inzetbare methodes die door meerdere anomalietests worden gebruikt. De implementatie van deze methodes volgde eveneens vrijwel direct uit de algemeen inzetbare algoritmen uit het vorige hoofdstuk.

## 7.3 Dataflow

Op basis van de voorgaande beschrijving van de klassenstructuur volgt nu een beschrijving van de werking van het programma en de dataflow (zie Figuur 15). In de klasse Main wordt het programma gestart. Main roept de klasse Inlezen aan, de klasse die ervoor zorgt dat de KB kan worden ingelezen. De output van Inlezen wordt samengevoegd tot een declaratieset en een Business Logica met Rule Base. Vervolgens wordt in Main een Window_Interface (interface window) gecreëerd. De declaratieset en Business Logica worden doorgegeven aan het interface window. Een volledige beschrijving van de mogelijkheden van het interface window volgt later in dit hoofdstuk. De belangrijkste functie van het window is in elk geval de knop "Toon Anomalieën". Deze initieert een nieuwe Verifieer_KB die de verschillende anomaliecategorieën die de gebruiker wenst te testen als input meekrijgt. Verifieer_KB initieert vervolgens de tests voor al deze anomaliecategorieën en voert deze uit. Een aantal tests (regelextensietests en onvolledigheidtests) maakt gebruik van een speciale klasse Environments. Deze klasse bevat een methode die alle mogelijke environments op basis van de Input Set creëert. Elke test geeft een string terug aan Verifieer_KB die bestaat uit een melding met uitleg als de desbetreffende anomalie voorkwam in de KB of een lege string als de anomalie niet voorkwam. Al deze resultaten worden vervolgens verzameld in één lange string die de output vormt van Verifieer_KB. Deze output wordt doorgegeven aan interface window en vervolgens op het scherm getoond.




**Figuur 15: Dataflow-schema**

## 7.4 Ontwerpbeslissingen

Een groot deel van de implementatie van het prototype volgde rechtstreeks uit de beschrijvingen van het model en de algoritmen. Andere zaken vergden echter meer denkwerk en soms moesten belangrijke keuzes worden gemaakt. In deze paragraaf wordt een overzicht gegeven van de belangrijkste ontwerpbeslissingen.

### 7.4.1 Vermijden van overlap tussen de anomalietests.

De definities van de verschillende categorieën van anomalieën in hoofdstuk 2, paragraaf 2.2.2 zijn zodanig opgesteld dat deze zoveel mogelijk exclusief zijn. Dit is niet zonder reden gedaan. De algoritmen en vervolgens de implementatie van alle anomalietests in het prototype, die rechtstreeks volgt uit deze definities, zijn daarom tevens exclusief ten opzichte van elkaar. Het grote voordeel hiervan is dat er geen dubbele of driedubbele meldingen komen voor dezelfde anomalie. Wanneer de test op contradictoire ketens bijvoorbeeld de test op contradictoire regelparen niet zou uitsluiten, en de test op contradictoire regelparen zou de test op contradictoire conclusies niet uitsluiten, dan zouden contradictoire conclusies door drie verschillende tests, op drie niveaus, worden gedetecteerd en vermeld. Het nadeel is echter wel dat zowel bepaalde definities, bepaalde algoritmen als tevens de code in bepaalde anomalieklassen van het prototype enigszins uitgebreider wordt dan zonder dit behoud van exclusiviteit.

### 7.4.2 Vaststellen van environments

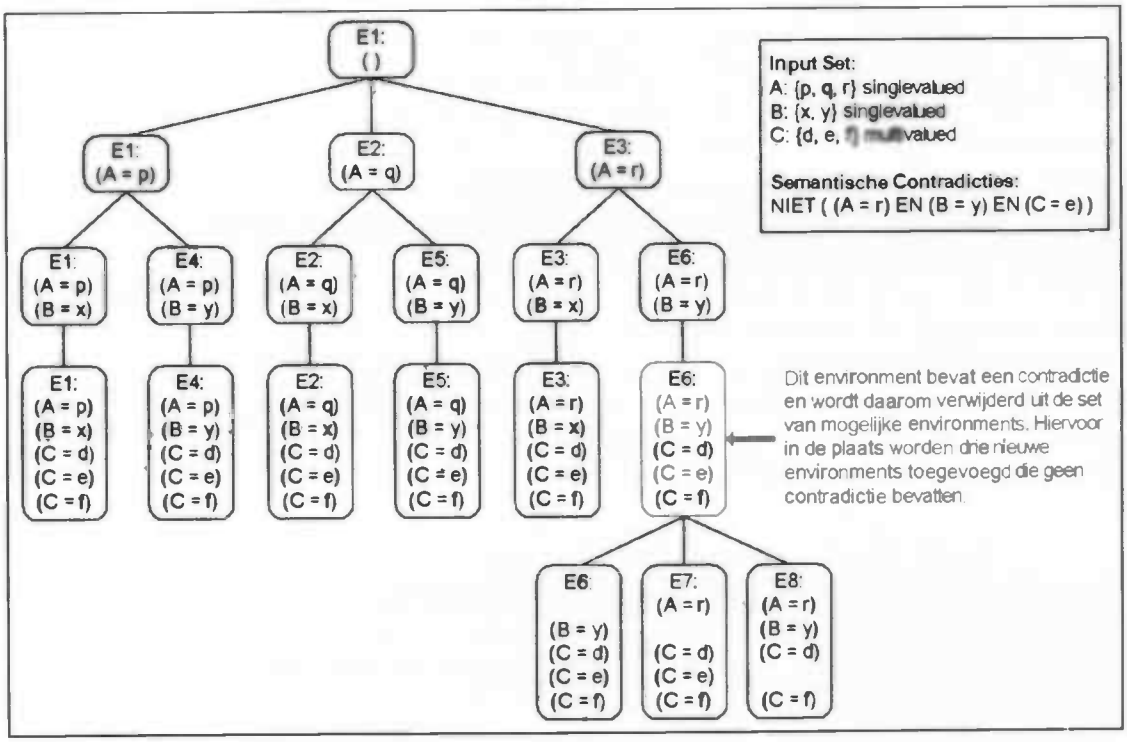
De tests voor contradictoire keten, circulaire keten, subsumed regel door keten, onbruikbare output en ontbrekende output maken gebruik van regelextensies op basis van alle mogelijke environments. Om alle environments te creëren, wordt per inputattribuut bekeken welke mogelijke waarden dit attribuut kan hebben. De basis voor het opbouwen van de environments is besproken in het vorige hoofdstuk, paragraaf 6.4. Tijdens de implementatie bleek echter dat er nog een aantal andere zaken aan de orde zijn bij het creëren van environments. In deze paragraaf wordt dieper ingegaan op deze complexiteiten.

In het geval van singlevalued inputattributen wordt voor elke mogelijke domeinwaarde een nieuw environment gecreëerd. In het geval van multivalued inputattributen gaat de opbouw van environments iets anders te werk. In dit geval kan het attribuut meerdere waarden tegelijk hebben. Theoretisch kan een environment dan bestaan uit slechts één literal van dit attribuut met een bepaalde waarde, maar het kan ook net zo goed zijn dat er een environment bestaat waarin alle domeinwaarden van het attribuut zijn vertegenwoordigd in literals.
Feitelijk zou elk mogelijk aantal en elke combinatie van domeinwaarden vertegenwoordigd moeten worden in de environments. Dit zou het aantal mogelijke environments echter nog sneller laten toenemen. Er is daarom gekozen voor een simplificatie op dit punt. Bij multivalued attributen worden gewoonweg alle domeinwaarden toegevoegd aan hetzelfde environment. Deze tactiek levert naar alle

waarschijnlijkheid de meest uitgebreide regelextensies op*, en daarmee dus de meeste kans op anomaliedetectie.

In de mogelijke environments mogen geen semantische contradicties voorkomen. Daarom wordt tijdens het opbouwen van de boom bij elke uitbreiding getest of er semantische contradictie is ontstaan. Als dit het geval is, dan wordt het gehele environment verwijderd uit de set van mogelijke environments. Vervolgens wordt dit verwijderde environment apart genomen en wordt er per inputliteral getest of het verwijderen ervan eveneens de semantische contradictie verwijderd. Wanneer dit zo is, dan is een environment zonder dit literal wel een mogelijk environment en zal deze worden toegevoegd aan de lijst van mogelijke environments.

Figuur 16 verduidelijkt het één en ander wat zojuist besproken is. In deze figuur is te zien hoe de boom van mogelijke environments wordt opgebouwd op basis van de inputattributen A, B en C en een semantische contradictie.




**Figuur 16: Creatie van mogelijke environments**

* *Via deze tactiek, worden in ieder geval de meest 'uitgebreide' input situaties vertegenwoordigd in de environments. Regels met antecedenten in de vorm:* ALS (voorkeur = blauw) DAN ... , *maar ook met antecedenten in de vorm:* ALS (voorkeur = groen) EN (voorkeur = blauw) EN (voorkeur = paars) DAN ... *waarbij voorkeur een multivalued attribuut is, worden door deze tactiek allemaal vuurbaar.*

### 7.4.3 Reduceren van de output van bepaalde tests

De tests op contradictoire en circulaire ketens testen of in één van de mogelijke environments een circulaire of contradictoire keten voorkomt. Echter, dezelfde contradictoire of circulaire keten kan in zeer veel verschillende environments voorkomen. Een zelfde probleem is aan de orde bij de detectie van ontbrekende output. Wanneer bepaalde regels ontbreken of verkeerd zijn geformuleerd, kan dit leiden tot dezelfde ontbrekende output in zeer veel verschillende environments. Om te voorkomen dat dezelfde anomalie, alleen dan telkens in een ander environment, zeer vaak wordt vermeld in de rapportage, is er voor gekozen om alleen melding te geven van het eerste voorkomen. Dit maakt voor de gebruiker in principe niet uit. Eén melding met het bijbehorende environment is in principe voldoende informatie om vast te stellen hoe de anomalie tot stand komt en hoe deze op te lossen.

### 7.4.4 Vaststellen van de regelketen uit de regelextensie

Een ander punt met betrekking tot de output van contradictoire en circulaire ketens betreft de vermelding van de regelketen die de anomalie veroorzaakt. De detectie van contradictie en circulariteit in een keten van regels vindt plaats via het creëren van een regelextensie (zie ter herinnering paragraaf 6.4). Bij het creëren van een regelextensie is het niet zo dat er een keuze wordt gemaakt in het vuren van de éne regel of de andere, maar *alle* regels die gevuurd kunnen worden, worden ook meteen gevuurd. Dit is het verschil tussen een regelketen en een regelextensie. Het gevolg van deze tactiek is dat tussen alle regels van de extensie ook regels kunnen voorkomen die niet nodig zijn voor het totstandkomen van de desbetreffende anomalie. Om een correcte melding te kunnen geven wordt eerst vastgesteld op welke punt de circulariteit of contradictie begint en op welk punt deze eindigt. Vervolgens worden uit deze sub-extensie de overbodige regels verwijderd. Hiervoor wordt een speciale methode gebruikt genaamd: ketenUitExtensie.

### 7.4.5 Complexiteit van conclusieliterals

In het prototype bestaan zowel het antecedent als de conclusie van een regel uit een verzameling literals. Dit betekent dat conclusieliterals ook een negatieve valentie kunnen hebben en bovendien de relatie >, <, >= of <= kunnen bevatten. Dat is immers voor alle literals mogelijk. Echter, in het Knowledge Framework van Everest zijn beide situaties niet mogelijk. Hoewel deze situaties niet foutief of onlogisch zijn, hebben de ontwerpers van het Knowledge Framework besloten dat het concluderen van een negatief literal of een waardebereik in de praktijk niet voorkomt. In principe is het prototype op dit vlak dus uitgebreider dan noodzakelijk. Het was echter extra werk geweest om deze mogelijkheid te verwijderen. Bovendien kan het geen kwaad en is het prototype zo alvast uitgerust voor een eventuele uitbreiding van het Knowledge Framework wat dit aspect betreft.

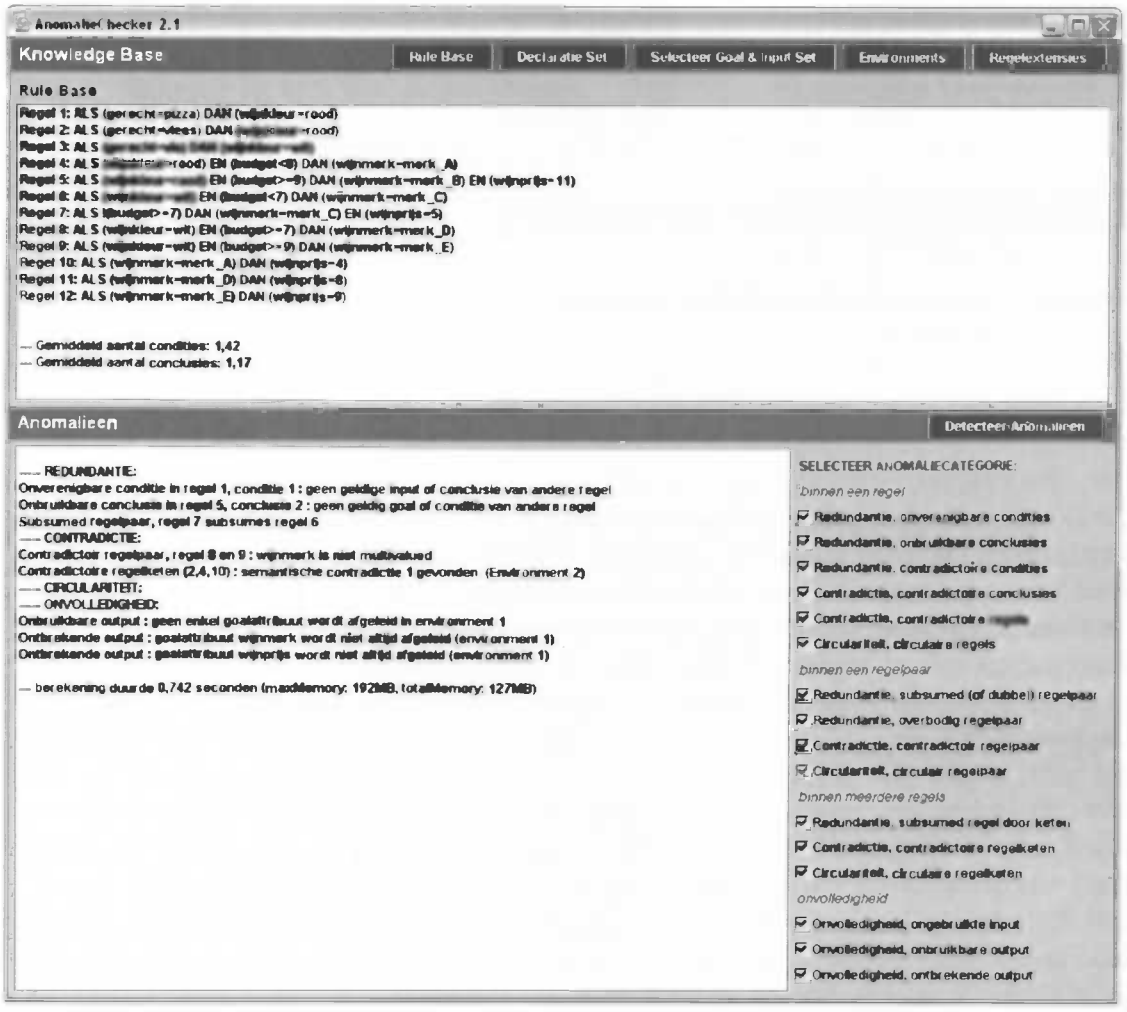
## 7.5 Gebruikersinterface

De gebruikersinterface is eenvoudig van opzet, aangezien het hier slechts om een prototype gaat. De belangrijkste functies van de gebruikersinterface betreffen:

- invoer van de Rule Base en de declaratieset
- tonen van alle regels uit de Rule Base op het scherm
- tonen van alle onderdelen van de declaratieset op het scherm
- een scherm waarin inputattributen en goalattributen gekozen kunnen worden
- tonen van de mogelijke environments op basis van de input set
- tonen van de regelextensies voor ieder environment
- onafhankelijk kiezen van alle anomaliecategorieën (via checkboxes)
- rapportage van alle gevonden anomalieën

### 7.5.1 Schermen

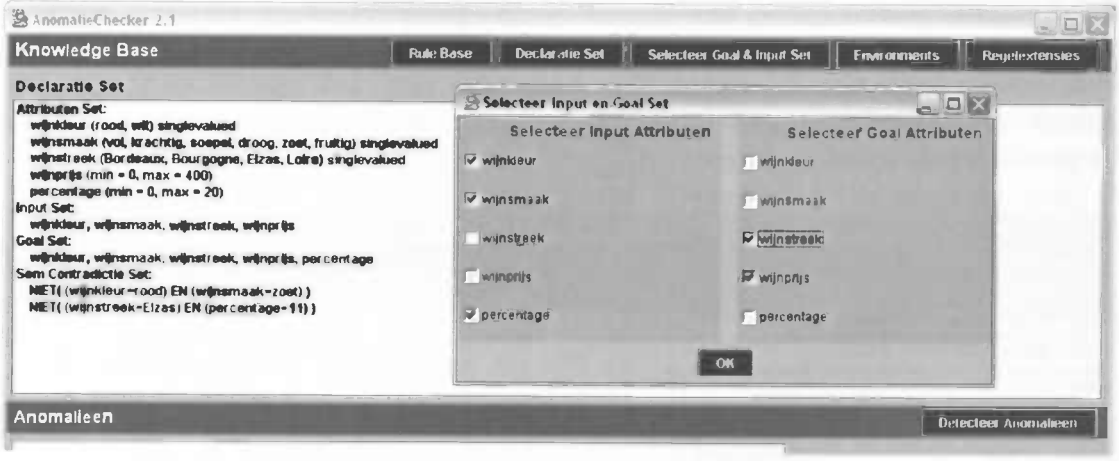
Het schermvoorbeeld in Figuur 17 laat zien hoe het hoofdscherm van de gebruikersinterface er uit ziet. In dit voorbeeld is reeds de rapportage aanwezig voor de geselecteerde anomalieën.




**Figuur 17: Schermvoorbeeld van de gebruikersinterface**

Het bovenste venster heeft betrekking op de Knowledge Base. In dit venster wordt de Rule Base getoond, de declaratieset, de mogelijke environments en de regelextensies voor elk environment. Het onderste venster heeft betrekking op de anomalieën in de KB. Hier kunnen de verschillende typen van anomalieën onafhankelijk van elkaar worden geselecteerd en vervolgens getest door te klikken op de knop 'Detecteer Anomalieën'. In het venster verschijnt dan een uitgebreide rapportage van alle gevonden anomalieën.

In Figuur 18 is te zien hoe de gebruiker in staat wordt gesteld te selecteren welke attributen hij als inputattribuut beschouwd en welke als goalattribuut. Op deze wijze kan de volledige Input Set en Goal Set geselecteerd worden. Echter, er kan ook een specifieke testsituatie gecreëerd worden met slechts een bepaald aantal input- en goalattributen. Zo kan de meest uitgebreide test op onvolledigheid, ontbrekende output, worden gedaan.




**Figuur 18: Selectie van Input Set en Goal Set**

## 7.5.2 Invoer

Om het prototype gemakkelijker te kunnen testen kunnen de Rule Base en de declaratieset worden ingelezen vanuit een tekstbestand. Speciaal hiervoor is de klasse Inlezen aangemaakt. Via de techniek van reguliere expressies worden literals, regels, attributen, de input set, de goal set en semantische contradicties in het tekstbestand herkend en correct ingevoerd. Het is hierbij van belang dat de invoer aan een bepaald standaard formaat voldoet. Enkele voorbeelden van deze tekstbestanden zijn te zien in Appendix D.

# H8 TESTOPZET & RESULTATEN

In dit hoofdstuk zal beschreven worden op welke wijze het prototype is getest, waarom er gekozen is voor bepaalde testmethoden en welke resultaten er zijn behaald met het prototype.

## 8.1 Testopzet

In hoofdstuk 5, paragraaf 5.2 zijn de evaluatiecriteria vastgesteld. Deze criteria kunnen worden verdeeld in twee hoofdcriteria:

1. **Correctheid van de detectie**
   De volledigheid van het prototype met betrekking tot alle categorieën en subcategorieën van anomalieën evenals het aandeel correcte detectie en het eventuele aandeel foutieve detectie (false positives).

2. **Bruikbaarheid in de praktijk**
   De snelheid van de anomaliedetectie in verschillende, praktisch reële situaties.

Ten einde het eerste hoofdcriterium te kunnen testen zijn er een drietal test-KB's gemaakt. Er is één test-KB gemaakt speciaal voor het testen van simpele categorieën van anomalieën (KB-A), één voor het testen van complexere anomalieën (KB-B) en één waarin geen anomalieën voorkwamen (KB-C). De test-KB's zijn in te zien in Appendix D. Deze drie test-KB's zijn allen niet langer dan ongeveer 30 regels. Ze zijn dus aanzienlijk kleiner dan gemiddelde KB's uit de praktijk. Er zijn echter belangrijke redenen voor het feit dat gekozen is voor kleine test-KB's. Ten eerste maakt het voor het evalueren van de correctheid van de detectie in principe niet uit hoe groot de KB is. Kleine test-KB's zijn dan handiger aangezien ze zelf bedacht en geconstrueerd moeten worden.* Ten tweede, en dat is nog veel belangrijker, is er een belangrijk probleem met het testen van het prototype bij grotere, complexere KB's. Dit gaat om de vraag: hoe weet je zeker of een gedetecteerde anomalie ook daadwerkelijk een anomalie is? En hoe weet je zeker of er niet nog meer anomalieën zijn dan er worden gedetecteerd? Bij betrekkelijk kleine, overzichtelijke KB's is een 'handmatige' controle nog te doen. Maar naarmate KB's complexer worden, wordt de 'verificatie van de verificatie' een steeds groter probleem. Wat betreft het testen van correctheid zijn compacte KB's dus handiger.

Wat betreft het tweede hoofdcriterium, bruikbaarheid, gaat het vooral om de snelheid waarmee de verschillende tests kunnen worden uitgevoerd. De output van het prototype is daarom uitgebreid met een melding van de tijdsduur van de detectie. Wat betreft de bruikbaarheid is het juist wel van belang test-KB's te gebruiken die qua grootte en complexiteit overeenkomen met gemiddelde KB's uit de praktijk. Deze waren helaas niet voor handen. Het handmatig construeren van een dergelijke KB was onhaalbaar en onzinvol met betrekking tot de beperkte tijd die beschikbaar was voor dit project. Om toch iets te kunnen zeggen over de bruikbaarheid bij grotere en

* *Er bestaan helaas geen kant en klare test-KB's met anomalieën. Mochten ze wel bestaan, dan heb ik ze in ieder geval niet kunnen vinden op het internet.*

complexere KB's zijn daarom een aantal trucs gebruikt. Meer hierover volgt in paragraaf 8.3.

## 8.2 Resultaten: correctheid van de detectie

Tabel 12 geeft een overzicht van de eigenschappen van de test-KB's en de resultaten van de anomaliedetectie.Voor elke geteste categorie is aangegeven hoeveel anomalieën er daadwerkelijk aanwezig waren en hoeveel hiervan er gedetecteerd zijn. De test-KB met simpele anomalieën (KB-A) bevatte overigens ook veel complexe anomalieën (die voor een groot deel ontstonden vanwege de simpele anomalieën), maar hierop is niet getest bij deze KB.

| *Test-KB* | KB-A | KB-B | KB-C |
|---|---|---|---|
| **Regels** | 20 | 24 | 31 |
| **Attributen** | 5 | 7 | 7 |
| **Inputattributen** | 4 | 4 | 2 |
| ***Anomalieën*** | *(gedetecteerd / aanwezig)* | | |
| **Binnen een regel** | | | |
| Onverenigbare condities | 2/2 | 0/0 | 0/0 |
| Onbruikbare conclusies | 1/1 | 0/0 | 0/0 |
| Contradictoire condities | 2/2 | 0/0 | 0/0 |
| Contradictoire conclusies | 2/2 | 0/0 | 0/0 |
| Contradictoire regels | 2/2 | 0/0 | 0/0 |
| Circulaire regel of kolom | 2/2 | 0/0 | 0/0 |
| **Binnen een regelpaar** | | | |
| Subsumed/dubbel regelpaar | 2/2 | 0/0 | 0/0 |
| Overbodig regelpaar | 1/1 | 0/0 | 0/0 |
| Contradictoir regelpaar | 2/2 | 0/0 | 0/0 |
| Circulair regelpaar | 7/7 | 0/0 | 0/0 |
| **Binnen meerdere regels** | | | |
| Subsumed regel door keten | - | 1/1 | 0/0 |
| Contradictoire regelketen* | - | 1/2 | 0/0 |
| Circulaire regelketen* | - | 1/1 | 0/0 |
| **Onvolledigheid** | | | |
| Ongebruikte input | - | 1/1 | 0/0 |
| Onbruikbare output | - | 12/12 | 0/0 |
| Ontbrekende output* | - | 2/2 | 0/0 |

**Tabel 12: Resultaten met betrekking tot correctheid**

Uit Tabel 12 blijkt dat in slechts één geval een aanwezige anomalie niet werd gedetecteerd. Het ging hierbij om een semantische contradictie die tot uiting kwam in een regelketen. Deze semantische contradictie bestond uit meer dan twee literals en bovendien waren deze literals verspreid over meer dan twee regels in de regelketen. Figuur 18 geeft deze situatie aan.

In geen van de gevallen waren er meldingen van anomalieën die in werkelijkheid geen anomalie waren (false positives).

*Semantische Contradictie:*
(CPU_MHz<800) EN (besturingssysteem='win_98') EN (geschikte_combinatie='ja')

*Regelketen:*
ALS (CPU_MHz<800) EN (geheugen_MB<256) DAN (systeem='langzaam')
ALS (systeem='langzaam') DAN (besturingssysteem='win_98')
ALS (besturingssysteem='win_98') EN (gewenst_spel='Warcraft') DAN (geschikte_combinatie='ja')

**Figuur 18: Contradictoire keten die niet werd gedetecteerd**

* *Zoals reeds vermeld in paragraaf 7.4.3 wordt een circulaire of contradictoire keten die in meerdere environments voorkomt slechts éénmaal vermeld in de rapportage teneinde veel dubbele meldingen te voorkomen. Ook wanneer dezelfde ontbrekende output in meerdere environments voorkomt, wordt hiervan slechts éénmaal melding gegeven. Wanneer sprake is van voorkomens van dezelfde anomalie in meerdere environments, wordt dit beschouwd als één anomalie.*

## 8.3 Resultaten: bruikbaarheid

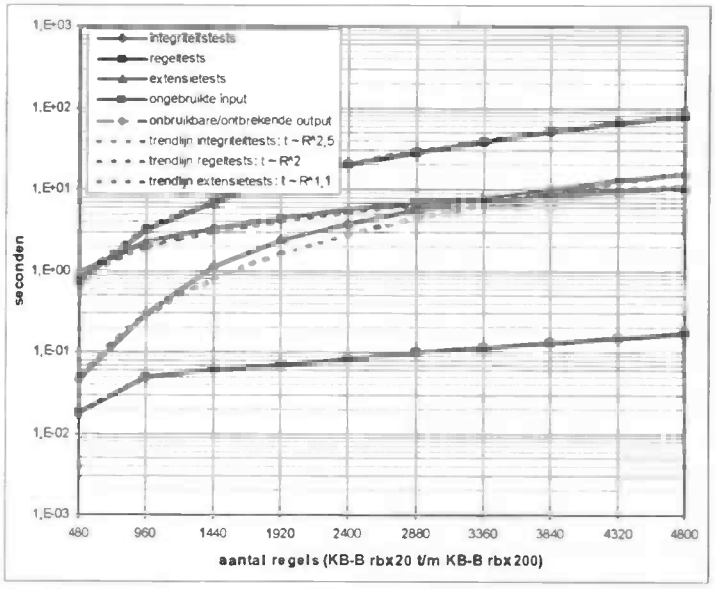
Om de snelheid van de detectie in grotere, complexere KB's te kunnen meten, zijn er aantal extra test-KB's geconstrueerd op basis van KB-B en KB-C. Op basis van KB-B zijn er test-KB's geconstrueerd waarin de originele Rule Base een aantal keer gekopieerd is. Dezelfde regels zijn dus meerdere malen opgenomen. Op deze wijze kon de performance worden gemeten bij KB's met veel regels. Door deze truc ontstaan natuurlijk wel veel dubbele regels.
Op basis van KB-C zijn test-KB's geconstrueerd door niet alleen de regels te kopiëren, maar door de gehele KB te kopiëren en vervolgens alle voorkomens van attribuutnamen in de kopie te wijzigen in nieuwe attribuutnamen. Niet alleen het aantal regels wordt zo vergroot, maar ook de attributenset en de inputset. Bovendien ontstaan er zo geen dubbele regels, elke regel is uniek. De zo ontstane KB-C A2 tot en met KB-C A6 bevatten een oplopend aantal inputattributen (2 tot en met 6). Deze KB's bevatten (net als KB-C) geen anomalieën, maar zijn wel heel geschikt voor het meten van de performance bij complexere KB's waarbij het aantal environments groot is. De trucs die hier zijn gebruikt leveren qua inhoud weliswaar geen realistische KB's op, maar voor testdoeleinden zijn ze wel degelijk nuttig. De eigenschappen van de gebruikte test-KB's staan in Tabel 13.

| | KB-B | | | | KB-C | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | rbx20 | rbx40 | ... | rbx200 | A2 | A3 | A4 | A5 | A6 |
| regels | 480 | 960 | ... | 4800 | 31 | 62 | 62 | 93 | 93 |
| gem. condities per regel | 2,38 | 2,38 | ... | 2,38 | 1,48 | 1,48 | 1,48 | 1,48 | 1,48 |
| gem. conclusies per regel | 1,00 | 1,00 | ... | 1,00 | 1,23 | 1,23 | 1,23 | 1,23 | 1,23 |
| attributen | 7 | 7 | ... | 7 | 7 | 14 | 14 | 21 | 21 |
| inputattributen | 4 | 4 | ... | 4 | 2 | 3 | 4 | 5 | 6 |
| environments | 144 | 144 | ... | 144 | 36 | 216 | 1296 | 7776 | 46656 |

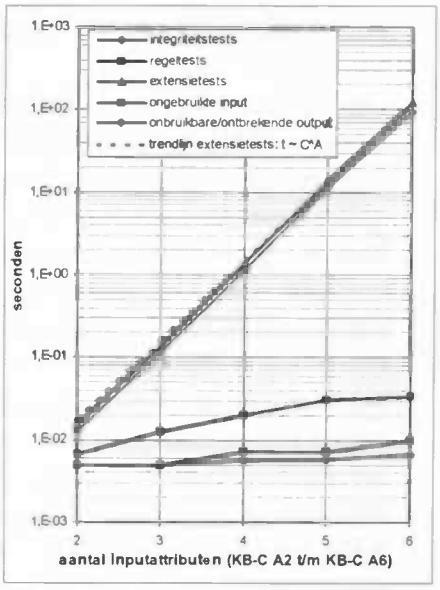
**Tabel 13: Eigenschappen test-KB's**

### 8.3.1 Snelheid anomaliedetectie

Figuur 18 en 19 geven de snelheid van de verschillende anomalietests aan voor de test-KB's KB-B rbx20 tot en met KB-B rbx200 en KB-C A2 tot en met KB-C A6. Alle tests zijn gedaan op een Pentium 3 op 1000 MHz en 256 MB intern geheugen. De snelheid van detectie is aangegeven op de verticale as. Merk op dat deze een logaritmische schaalverdeling heeft. Tests van hetzelfde type zijn gemiddeld en door één lijn weergegeven. Voor de losse gegevens wordt de lezer verwezen naar Appendix E. De grafiek van de extensietests geeft alleen het gemiddelde van de test op contradictoire ketens en op circulaire ketens. Niet meegenomen hierin is dus de test op redundantie in een keten. Deze test kostte namelijk dermate veel tijd, dat de test niet meer uitvoerbaar bleek voor het grootste deel van de test-KB's (zie wederom de originele, afzonderlijke gegevens in Appendix E). In hoofdstuk 10, Discussie en Aanbevelingen, zal verder ingegaan worden op de traagheid van deze specifieke test.




Figuur 19




Figuur 20

Figuur 19 en 20 laten een aantal interessante zaken zien:

1. Integriteittests zouden volgens de theorie (zie paragraaf 2.3.2) een computationele complexiteit moeten hebben waarbij de belangrijkste component lineair toeneemt met het aantal regels. In Figuur 19 zien we echter dat de belangrijkste component helemaal niet lineair is. In Appendix E is te zien dat dit geheel wijten is aan de tests op onverenigbare condities en onbruikbare conclusies. De algoritmen van deze tests (zie Appendix C) geven een verklaring. Naast een lineaire component met het aantal regels bevatten deze tests ook een kwadratische component. Deze vormt bij grote Rule Bases uiteindelijk verreweg de belangrijkste component. Het lijkt er echter op dat de toename zelfs meer dan kwadratisch is (zie de trendlijn voor integriteittests die voldoet aan en macht van 2,5). Een verklaring hiervoor kan liggen in het geheugengebruik van het programma bij deze specifieke tests.*

2. Regeltests hebben volgens de theorie (zie paragraaf 2.3.2) een computationele complexiteit waarbij de belangrijkste component kwadratisch toeneemt met het aantal regels. De resultaten laten dit beeld eveneens zien. Dit blijkt vooral uit Figuur 19. In deze figuur is een trendlijn uitgezet met een kwadratische vergelijking die goed samenvalt met de resultaten.

3. Extensietests en de tests op onbruikbare of ontbrekende output duren in alle situaties ongeveer even lang (de lijnen vallen in beide figuren vrijwel samen). Deze tests zijn afhankelijk van de constructie van mogelijke environments. In Figuur 20 blijkt dat dit duidelijk de belangrijkste en beperkende factor is in de detectiesnelheid. De trendlijn voor extensietests in deze figuur voldoet aan de exponentiële vergelijking $t \sim c^a$. Dit komt goed overeen met de theorie uit paragraaf 6.4.1. Hierbij komt c overeen met het gemiddelde aantal

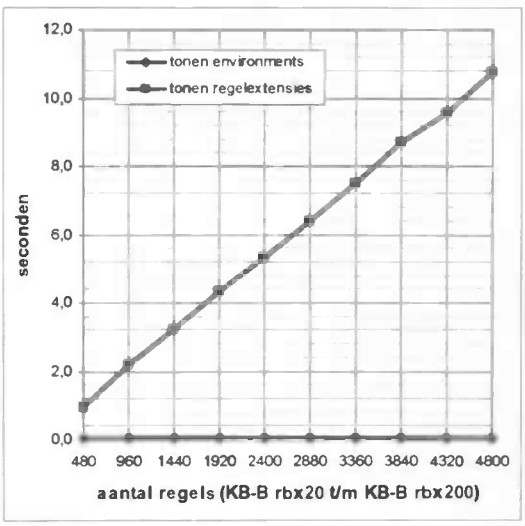
* *Het is mogelijk dat het programma vanaf een bepaalde drempel te weinig geheugen beschikbaar heeft. Dit zou een verloop kunnen verklaren dat uiteindelijk voldoet een kwadratische vergelijking, maar in het begin nog sneller dan kwadratisch toeneemt. Dit blijft echter slechts een hypothese. Extra onderzoek is nodig om een verklaring met meer zekerheid te kunnen geven.*

domeinwaarden van een inputattribuut en a staat voor het totale aantal inputattributen.

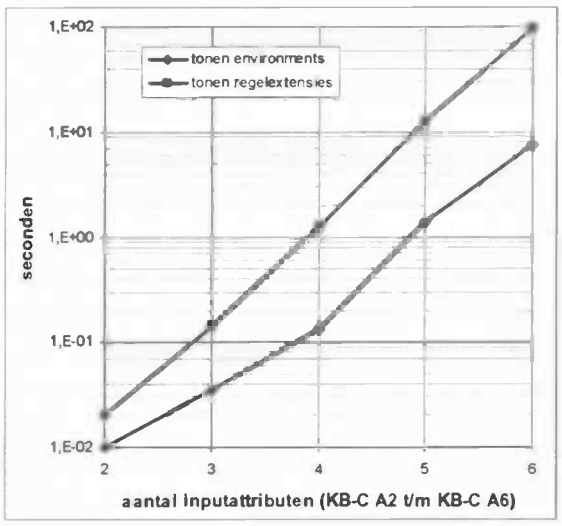
4. Bij de test-KB's in Figuur 19 is de hoeveelheid environments gering en is de beperkende factor juist de grootte van de Rule Base. Interessant om te zien is dat in deze situatie de detectiesnelheid van extensietests minder snel stijgt dan regeltests naarmate het aantal regels toeneemt. Een verklaring hiervoor ligt in de manier waarop regelextensies worden opgebouwd. Dit gaat volgens een WHILE-loop langs alle regels in de Rule Base. De loop stopt als er geen regels meer vuurbaar zijn. Over het algemeen zal de loop veel minder vaak worden uitgevoerd dan het totaal aantal regels. De complexiteit van de berekening is daarom een stuk minder dan kwadratisch ten opzichte van het aantal regels in de Rule Base. In Figuur 19 is een trendlijn uitgezet die goed samenvalt met de resultaten.

### 8.3.2 Vaststellen van environments en regelextensies

Figuur 21 en 22 laten zien hoe lang het tonen van de environments en de regelextensies duurt voor de test-KB's KB-B rbx20 tot en met KB-B rbx200 en KB-C A2 tot en met KB-C A6. De verticale as geeft hier de berekentijd in seconden aan (merk op dat deze een logaritmische schaal heeft in Figuur 22). Deze resultaten zijn in overeenstemming met de uitkomsten in de vorige paragraaf.
Zo is te zien in Figuur 21 dat de toename van duur van het vaststellen van de regelextensies naarmate het aantal regels toeneemt, in overeenstemming is met de resultaten van de tests op basis van regelextensies in Figuur 19. In Figuur 22 is duidelijk te zien is dat het vaststellen van de environments exponentieel toeneemt met het aantal inputattributen, vergelijkbaar met de toename in Figuur 20. Het vaststellen van de regelextensies is in dit geval sterk gerelateerd aan het vaststellen van de environments. Immers, de regelextensies worden op basis van environments gemaakt.




Figuur 21




Figuur 22

# H9 CONCLUSIES

## 9.1 Conclusies met betrekking tot correctheid

De doelstellingen op het gebied van correctheid waren:

- Alle anomaliecategorieën moeten gedetecteerd kunnen worden
- Verreweg het grootste deel van de testvoorbeelden van anomalieën moet kunnen worden gedetecteerd
- Er mogen geen of uiterst weinig false positives voorkomen in de anomaliedetectie

Uit de resultaten blijkt dat al deze doelstellingen behaald zijn. Alleen in het geval een contradictoire regelketen waarbij de literals van een semantische contradictie op meer dan twee plaatsen in de keten worden afgeleid, wordt de anomalie niet gedetecteerd. Dit is vanwege een verkeerd opgesteld algoritme. Het gaat hierbij echter om een zeer specifieke situatie. Bovendien is het algoritme naar alle waarschijnlijkheid eenvoudig aan te passen zodat het deze situatie wel kan detecteren. In zijn algemeenheid kan worden geconcludeerd dat de tool volledig is met betrekking tot alle anomaliecategorieën. Tevens zijn er geen meldingen van false positives geweest tijdens het testen.

## 9.2 Conclusies met betrekking tot bruikbaarheid

De doelstelling op het gebied van bruikbaarheid (hoofdstuk 5, paragraaf 5.2) was dat een gemiddelde KB in maximaal een kwartier moet kunnen worden getest op een gemiddelde computer. De gebruikte test-KB's voldeden geen van allen aan de eigenschappen van de gemiddelde KB zoals die is aangegeven bij deze doelstelling. Op basis van de resultaten van de test-KB's kunnen echter wel voorspellingen gedaan worden.

### 9.2.1 Integriteittests en regeltests

Het testen van deze categorieën toegepast op een test-KB van 4800 regels (KB-B rbx200) duurde slechts enkele tientallen seconden. Deze test-KB bevatte een lager gemiddeld aantal condities en conclusies per regel dan de gemiddelde KB. Echter, het gemiddeld aantal conclusies en condities van de gemiddelde KB ligt niet zo heel veel hoger, terwijl het aantal regels juist minder is dan 4800. Het is daarom aannemelijk dat integriteittests en regeltests bij een gemiddelde KB eveneens enkele tientallen seconden zullen duren. Daarmee is de tool voor deze categorieën snel genoeg.

### 9.2.2 Regelextensietests

Bij regelextensietests blijkt dat niet zo zeer de grootte van de Rule Base, maar de hoeveelheid environments de 'bottleneck' vormt in de snelheid van de detectie. Regelextensietests vinden immers plaats op basis van environments. Het is duidelijk dat de hoeveelheid environments exponentieel toeneemt met het aantal inputattributen. De gemiddelde KB uit de praktijk bevat zeer veel attributen waarvan

vele ook input kunnen zijn. De resultaten van een test-KB met 6 inputattributen resulteerde al in 46.656 mogelijke environments. De test op contradictoire of circulaire ketens nam hierbij gemiddeld al meer dan 100 seconden in beslag. Wanneer de stijgende lijn van Figuur 20 op pagina 58 zou worden voortgezet, dan zouden de extensietests bij een KB met 10 inputattributen $10^6$ seconden duren, oftewel bijna 12 dagen! Wat betreft de test op subsumed regels door een keten ligt de duur van de test nog een aantal factoren hoger dan de test op contradictoire en circulaire ketens. Er kan geconcludeerd worden dat het prototype wat regelextensietests betreft in de praktijk niet bruikbaar zal zijn. In het hoofdstuk Discussie & Aanbevelingen zullen eventuele oplossingen voor dit probleem worden aangedragen.

## 9.3 Vergelijking met andere tools

In hoofdstuk 3 is een aantal reeds bestaande tools onderzocht. Wanneer het prototype wordt vergeleken met deze onderzochte tools, dan kan worden gezegd dat het prototype op veel punten volledliger en uitgebreider is. Tabel 14 geeft een overzicht van dezelfde eigenschappen als Tabel 9 in hoofdstuk 3, paragraaf 3.5, ingevuld voor het prototype.

| | **Prototype** |
|---|---|
| **Type KB** | regels |
| **Platform** | stand alone, input als txt-file |
| **Syntax expressies** | propositionele deel predikatenlogica |
| **Relaties** | =, <, >, >=, <= |
| **Negatieve literals** | ja |
| **Sem. contradicties** | ja |
| **Goal Set & Input Set** | ja |
| **Single/multivalued attributen** | beide |
| **Heuristisch zoeken** | nee |
| **Volledige extensies** | ja |
| **Exclusief testen van alle categorieën** | ja |
| **Categorieën die *niet* worden getest** | geen |

**Tabel 14: Eigenschappen van het prototype**

# H10 DISCUSSIE & AANBEVELINGEN

Tijdens dit afstudeerproject is geen volledig werkzame verificatiemodule voor het Knowledge Framework gebouwd, maar is genoegen genomen met een prototype met beperktere complexiteit. Het prototype moet gezien worden als een soort pilot naar de mogelijkheden en haalbaarheid van een toekomstige implementatie van een verificatiemodule in het Knowledge Framework van Everest. Het prototype is op de meeste punten geslaagd, gegeven de doelstellingen voor dit afstudeerproject. Bovendien is het prototype op bepaalde punten reeds vollediger dan bestaande commerciële tools of tools uit de academische wereld (zie hoofdstuk 3). Dit wil echter niet zeggen dat het prototype hiermee 'af' is. Er zijn vele kleine maar ook grote verbeteringen aan te wijzen. In dit hoofdstuk zal een opsomming worden gegeven van deze eventuele toekomstige verbeteringen. Daarna zal nog een paragraaf worden besteed aan enkele aanbevelingen richting Everest. Sommige van deze aanbevelingen hebben direct toepassing op het prototype, andere juist op verificatie in bredere zin.

## 10.1 Verbeteren van de performance

Het is gebleken dat de tactiek van het opbouwen van mogelijke environments op zichzelf zeer bruikbaar is. De tactiek is bij meerdere anomalietests inzetbaar: contradictoire en circulaire ketens, subsumed regels door ketens, onbruikbare output en ontbrekende output. Het grote nadeel dat inmiddels duidelijk zal zijn, is de explosieve stijging van het aantal mogelijke environments waardoor de performance van deze tests in het geding komt. Een aantal mogelijke oplossingen voor dit probleem zullen nu worden behandeld.

1. **Bewaren van de environments voor eventuele andere tests.**
   Dit is de meest simpele manier om het prototype in zijn geheel aanzienlijk sneller te maken. De tests die afhankelijk zijn van environments maken allemaal dezelfde environments aan. Wanneer de environments slechts één keer hoeven worden aangemaakt en vervolgens kunnen worden hergebruikt, zal dit de algemene performance aanzienlijk verbeteren.

2. **Zorgvuldig kiezen van de inputattributen.**
   Alleen de attributen die noodzakelijke input zijn aan de Business Logica (de vereniging van alle beslistabellen en businessregels) zijn inputattributen.

3. **Modulair testen.**
   Het kan raadzaam zijn onderzoek te doen naar de structuur van de KB. Als bepaalde modules van beslistabellen en regels feitelijk los staan van andere modules, kunnen deze ook los worden getest. Alleen de attributen die input zijn aan die module moeten dan als Input Set worden geselecteerd en ook de Goal Set moet worden beperkt tot de goalattributen van alleen die module. Het is niet onwaarschijnlijk dat in de praktijk veel KB's inderdaad zijn te zien als losse, naast elkaar staande modules. Maar ook wanneer modules niet geheel los van elkaar staan is modulair testen wellicht mogelijk. Dit blijkt uit een artikel van Cornelissen, Jonker en Treur [22]. Zij hebben onderzoek gedaan naar de mogelijkheden van compositionele verificatie. Hierbij worden de structurele eigenschappen van de KB in kaart gebracht. Op basis daarvan kan een

hiërarchische decompositie van de KB in verschillende componenten plaatsvinden. De subcomponenten kunnen dan vervolgens worden geverifieerd. Hierbij is het wel noodzakelijk dat de subcomponenten geen relaties onderling mogen hebben.

4. **Slimmer opbouwen van de environments.**
   Bij attributen van het type string worden nu voor alle mogelijke domeinwaarden nieuwe environments gecreëerd. Achteraf gezien was het misschien slimmer geweest om dezelfde tactiek te gebruiken als bij attributen van het type integer. Dat wil zeggen dat alleen de inputliterals worden gecreëerd die worden gebruikt in de antecedenten en hun tegengestelden. Stel dat er bijvoorbeeld een attribuut 'kleur' met de domeinwaarden 'rood', 'geel', 'groen', 'blauw' en 'paars' bestaat. Stel vervolgens dat de Rule Base alleen de regels ALS (kleur = 'rood') DAN ... en ALS NIET(kleur = 'rood') DAN ... bevat, dan is het voldoende om alleen een environment te creëren met literal (kleur = 'rood') en één met het literal (kleur = 'geel'). Deze tactiek zorgt tevens voor een ander voordeel. Er is namelijk ook geen verschil meer in de tactiek tussen singlevalued en multivalued inputattributen. In beide gevallen zullen de gecreëerde environments volledig zijn. Dit in tegenstelling tot de 'oude' methode waarbij in het geval van multivalued inputattributen nog een simplificatie moest worden toegepast (zie paragraaf 7.4.2).

5. **Alleen een random selectie van environments**
   Natuurlijk is er altijd nog de mogelijkheid om gewoonweg niet alle mogelijke environments te creëren, maar slechts een random gekozen selectie van alle mogelijke environments.

6. **Gebruik van snellere hardware**
   De computer waarmee de test zijn uitgevoerd (Pentium 3, 1000 MHz, 256 MB RAM geheugen) was ten tijde van het onderzoek eigenlijk alweer verouderd. Er is zeker winst te behalen door het gebruik van snellere hardware.

## 10.2 Overige verbeteringen

**Uitbreiding van het aantal attribuuttypen**
Naast attributen van het type string en integer zou het prototype kunnen worden uitgebreid met alle mogelijk attribuuttypen uit het Knowledge Framework. Dit zal waarschijnlijk niet eens heel veel werk zijn. Zo is een attribuut van het type boolean te behandelen als een attribuut van het type string maar dan slechts met twee domeinwaarden ('true' en 'false'). Attributen van het type real, currency of date kunnen grotendeels net zo behandeld worden als attributen van het type integer.

**Uitbreiding van de expressiviteit van literalwaarden.**
De waarde van een literal mag in het prototype geen wiskundige expressie bevatten. In het Knowledge Framework is dit echter wel mogelijk (zie paragraaf 4.3.1). Het prototype zou dus moeten worden uitgebreid met een speciale module die wiskundige expressies kan uitrekenen of vergelijken.

**Aanvulling op de detectie van contradicties**
Wanneer de literals NIET(wijnkleur = 'rood'), NIET(wijnkleur = 'wit') en NIET(wijnkleur = 'rosé') allemaal zijn afgeleid, en er bestaan niet meer domeinwaarden dan 'rood', 'wit' en 'rosé', dan kan men spreken van een contradictie. Er is namelijk altijd één van de domeinwaarden geldig. Deze vorm van contradictie wordt echter niet gedetecteerd door het prototype.

**Semantische contradicties van meer dan drie literals**
Zoals gebleken is worden deze anomalieën niet gedetecteerd in een keten van regels wanneer de literals van de contradictie op meer dan twee verschillende plaatsen in de keten voorkomen. Het algoritme voor de detectie van contradictoire ketens (zie Appendix C) geeft hiervoor de verklaring: er worden telkens slechts twee regels uit de keten met elkaar vergeleken. Deze test moet dus worden verbeterd op dit punt.

**Traagheid van de detectie van subsumed regels door een keten**
Het is gebleken dat deze specifieke test nog vele factoren langzamer is dan de twee andere extensietests (circulaire en contradictoire keten). De test werkt dan ook geheel anders dan de twee andere extensietests (zie paragraaf 6.4.3). Er worden voor alle environments regelextensies gemaakt voor Rule Bases met en zonder de aanwezigheid van de regel. Het is te voorspellen dat de test dus in ieder geval minstens net zo lang duurt als het opbouwen van alle environments. Echter, aangezien de test nog vele malen langer duurt, moet er nog een veel grotere beperkende factor zijn. Meer onderzoek hiernaar is noodzakelijk.

**Detectie van redundante literals**
In het artikel van Du Zang en Luqi [9] wordt naast redundante regels ook gesproken over redundante literals. Het meest simpele voorbeeld van een redundant literal bevindt zich in de volgende regel: ALS A EN B EN A DAN C. Wellicht zijn er nog complexere vormen aan te wijzen van redundante literals. Er is hier echter geen onderzoek naar gedaan aangezien het hier om de minst schadelijke en in de praktijk waarschijnlijk ook om de minst voorkomende vorm van anomalieën gaat.

## 10.3 Aanbevelingen voor Everest

### 10.3.1 Anomaliedetectie in regelgebaseerde kennisrepresentatie

Tijdens dit afstudeerproject is ervoor gekozen om de algoritmiek en het prototype te construeren voor de verificatie van kennissystemen op basis van regels. Verificatietechnieken die direct opereren op beslistabelgebaseerde systemen zijn buiten beschouwing gelaten. Deze keuze is bewust gemaakt op basis van het feit dat bijna alle categorieën van anomalieën gemakkelijker zijn te detecteren in een regelgebaseerde structuur (zie hoofdstuk 4, paragraaf 4.3). Everest wordt geadviseerd om deze ingeslagen weg, dat wil zeggen, anomaliedetectie op basis van een regelstructuur, te blijven volgen. Ten einde beslistabellen te verifiëren wordt aangeraden deze om te zetten in regels. Hoewel er tijdens dit afstudeeronderzoek geen onderzoek naar gedaan is, is de verwachting dat dit gemakkelijk te realiseren zal zijn. Immers, het prototype accepteert reeds regels met zowel meerdere condities als meerdere conclusies. Dergelijke regels zijn in feite al gelijk aan de losse kolommen

**Beslistabel:**

| kleur | rood | | | blauw | | groen |
|---|---|---|---|---|---|---|
| type | A | B | C | A | [] | * |
| prijs | 2 | 3 | 4 | 5 | 6 | 7 |

**Omzetting in regels:**
ALS (kleur = 'rood') EN (type = 'A') DAN (prijs = 2)
ALS (kleur = 'rood') EN (type = 'B') DAN (prijs = 3)
ALS (kleur = 'rood') EN (type = 'C') DAN (prijs = 4)
ALS (kleur = 'blauw') EN (type = 'A') DAN (prijs = 5)
ALS (kleur = 'blauw') EN NIET(type = 'A') DAN (prijs = 6)
ALS (kleur = 'groen') DAN (prijs = 7)

**Figuur 23: Omzetting beslistabel naar regels**

van expanded beslistabellen. Zie de definities van beslistabel-gebaseerde systemen in paragraaf 2.1.2. Wanneer sprake is van contracted en/of limited beslistabellen is de omzetting naar regels iets complexer, maar nog steeds goed mogelijk. Het voorbeeld in Figuur 23 laat dit zien.

## 10.3.2 Teststrategie

Hoewel het in principe mogelijk is om alle anomalietests in één keer uit te voeren, zal het in de meeste situaties verstandiger zijn om een bepaalde volgorde van testen aan te houden. Hiermee wordt bedoeld dat de anomalieën die eventueel worden gedetecteerd door minder complexe tests eerst worden opgelost, alvorens men begint met complexere anomalietests. Deze tactiek heeft een aantal voordelen.

Ten eerste is het zo dat in de praktijk de meest voorkomende anomalieën kunnen worden gedetecteerd met integriteittests en regeltests. Althans, dit is wat ervaren kennistechnologen van Everest aangeven (zie paragraaf 4.2). De anomalieën die alleen met regelextensietests kunnen worden gedetecteerd vormen dus slechts een klein deel van het totale aantal te verwachten anomalieën. Hier staat tegenover dat regelextensietests wel veel meer tijd kosten vanwege het testen in alle mogelijke environments. Dit kan behoorlijk oplopen zoals we gezien hebben in de resultaten van de test-KB's met veel inputattributen. Door de regelextensietests in eerste instantie achterwege te laten kan men een groot gedeelte van de anomalieën detecteren (en oplossen), terwijl de detectie slechts weinig tijd in beslag neemt.
Ten tweede is er nog een ander probleem dat tijdens het testen van het prototype naar voren is gekomen. Het blijkt dat eenvoudige anomalieën soms de oorzaak kunnen vormen van complexere anomalieën. Eén anomalie kan meerdere andere anomalieën teweeg brengen. Een voorbeeld daarvan is te zien in Figuur 24. Ook wat dit betreft is het dus verstandig om niet alle tests in één keer uit te voeren, maar om te beginnen met alleen de eenvoudige tests, vervolgens de gevonden anomalieën te repareren, en pas daarna complexere tests uit te voeren

| | | |
|---|---|---|
| ALS (type=X) DAN (kluer=rood) | onbruikbare conclusie | *originele anomalie* |
| ALS (kleur=rood) DAN (prijs=12) | onverenigbare condities en ontbrekende of onbruikbare output | *extra anomalieën als gevolg* |

**Figuur 24: Oorzaak van anomalieën**

Een derde probleem heeft te maken met de manier waarop contradicties worden gedetecteerd. Alle tests zijn zodanig geïmplementeerd dat ze exclusief zijn ten opzichte van elkaar. Zo worden bijvoorbeeld alleen contradictoire regels gerapporteerd, wanneer het antecedent en de conclusie van de regel geen contradicties bevatten, maar de samenstelling van het antecedent en de conclusie bevat wel contradicties. Het gevaar dat hierin schuilt, is dat er ook situaties denkbaar zijn

waarbij bijvoorbeeld de conclusie van een regel weliswaar een contradictie bevat, maar dat de regel als geheel ook een (andere) contradictie bevat. Deze laatste contradictie wordt pas gedetecteerd wanneer de eerste contradictie is opgelost door de gebruiker. Een soortgelijk probleem is aan de orde bij de detectie van contradictoire regelparen en contradictoire ketens.

# NAWOORD

*How often have I said to you that when you have eliminated the impossible, whatever remains, however improbable, must be the truth?*

Sir Arthur Conan Doyle *(Sherlock Holmes)*

Deze wijsheid van Sherlock Holmes refereert in de eerste plaats natuurlijk aan het oplossen van misdaden. Echter, ze vormt net zo goed de kern van het verifiëren van een kennissysteem! Immers, na het verwijderen of repareren van alle anomalieën, houdt men het 'ware' kennissysteem over.

Ik heb getracht tijdens mijn afstudeerproject een zo uitgebreid mogelijk beeld te vormen van alle soorten anomalieën die kunnen voorkomen in een kennissysteem. Ik heb bovendien een prototype geconstrueerd dat bewijst dat detectie van al die verschillende soorten van anomalieën in de praktijk mogelijk is. Mijn onderzoek is hierin uitgebreider en vollediger geweest dan voorgaande studies waar ik over gelezen heb.

Dat wil echter niet zeggen dat hiermee de kous af is. Zo moet er bijvoorbeeld nog gewerkt worden aan de snelheid van detectie in praktijksituaties. Tevens zal er nog veel werk verricht moeten worden om de verificatie mogelijk te maken wanneer alle details van het Knowledge Framework van Everest aan de orde zijn. Maar dit verslag en het prototype vormen een goede basis waarop voortgebouwd kan worden.

Zes maanden lang heb ik met veel interesse en enthousiasme gewerkt aan dit onderzoek. Graag wil ik Rineke Verbrugge, Mark Mastop en Onno de Groote hartelijk danken voor hun begeleiding, adviezen, vertrouwen en een prettige samenwerking.

# LITERATUUR


[1] S. Smith, A. Kandel. (1993). Verification and Validation of Rule-Based Expert Systems. *CRC Press Inc.*

[2] A.D. Preece, R. Shinghal. (1994). Foundations and applications of Knowledge Base Verification. *International Journal of Intelligent Systems, Vol 9, 683-701.* http://citeseer.ist.psu.edu/preece94foundation.html

[3] A. Preece, S. Talbot, L. Vignollet. (1997). Evaluation of Verification Tools for Knowledge-Based Systems. *International Journal of Human-Computer Studies, 47, 629-658.* http://www.csd.abdn.ac.uk/~apreece/research/download/ijhcs1997.pdf

[4] A. D. Preece. (1998). Building the Right System Right Evaluating V&V Methods in Knowledge Engineering. *Proceedings of KAW '98.*

[5] J. Vanthienen. (2001). Ruling the Business: About Business Rules and Decision tables. *Vandenbulcke, J. & Snoeck, M. (eds.): New Directions in Software Engineering, Leuven University Press, Leuven, pp. 103–120.*

[6] J. Vanthienen, A. Aerts, C. Mues, G. Wets. (1995). A Modelling Approach to Knowledge Based Systems Verification. *11th Conference on AI for Applications (IEEE CAIA '95), Verification and Validation Workshop, Los Angeles, February 19-22.*

[7] J. Vanthienen, C. Mues, G. Wets, K. Delaere. (1998). A tool-supported approach to inter-tabular verification. *Expert Systems with Applications, 15, pp. 277-285.*

[8] A.P. Eremeev. (2001). On Correctness of the Production Decision Model Based on the Decision Tables. *Automation-and-Remote-Control. Oct. 2001; 62(10): 1608-19; Original: Avtomatika-i-Telemekhanika. Oct. 2001; (10): 78-90.*

[9] Du Zang, Luqi. (1999). Approximate declarative semantics for rule base anomalies. *Knowledge-Based Systems, Vol. 12, No. 7, pages 341-353.*

[10] Frank van Harmelen. (1998). Applying rule-base anomalies to KADS inference structures. *Decision Support Systems, vol. 21, no. 4, pp. 271--280.* http://citeseer.ist.psu.edu/vanharmelen98applying.html

[11] S. Spreeuwenberg, R. Gerrits, & M. Boekenoogen. (2000). VALENS: A Knowledge Based Tool to Validate and Verify an Aion Knowledge base. *Proceedings of the PAIS 2000 Conference (ECAI 2000)* http://citeseer.ist.psu.edu/spreeuwenberg00valens.html

[12] S. Spreeuwenberg, R. Gerrits. (2002). Requirements for successful verification in practice. *Proceedings of FLAIRS-2002: The 15th International FLAIRS Conference*

[13] S. Spreeuwenberg. (2003). Using Verification and Validation Techniques for High-quality Business Rules. *Business Rules Journal, Vol. 4, No. 2, (Feb. 2003)* http://www.BRCommunity.com/a2003/b132.html

[14] R. Gerrits. (2003). Verification of Business Rules Utilization. *Business Rules Journal, Vol. 4, No. 12 (Dec. 2003)* http://www.BRCommunity.com/a2003/b175.html

[15] J. Santos, C. Ramos, Z. Vale, M. Marques. (1999). VERITAS - An Application for Knowledge Verification. *ICTAI, p441.* http://citeseer.ist.psu.edu/688930.html

[16] S. Murell, R. Plant. (1997). A survey of tools for the validation and verification of KBS: 1985-1995. *Decision-Support-Systems. Dec. 1997; 21(4): 307-23.*

[17] M. Willems, R. Verbrugge. (1996). Logische Technieken in de AI. *Dictaat.*

[18] R. F. Gamble, D. M. A. Baughman. (1996). A methodology to incorporate formal methods in hybrid KBS verification. *International Journal of Human-Computer Studies, v.44 n.2, p.213-244, Feb. 1996.*

[19] R. F. Gamble, R. Mukherjee, J. A. Parkinson. (1997). Classifying and detecting anomalies in hybrid knowledge-based systems. *Decision Support Systems, December 1997, vol. 21, no. 4, pp. 231-251.*

[20] L. Hermans, S. Spreeuwenberg. (2004). De essentie van de "Business Rules Aanpak". *In bewerking.*

[21] T. Van der Weij. (1994). Integriteitscontroles voor beslissingstabellen in AKTS. *Stageverslag.*

[22] F. Cornelissen, C. M. Jonker, J. Treur. (1997). Compositional Verification of Knowledge-Based Systems: a Case Study for Diagnostic Reasoning. *In: Proc. European Knowledge Acquisition Workshop EKAW'97, Springer-Verlag*
http://citeseer.ist.psu.edu/cornelissen97compositional.html.

# APPENDIX A

## Extra definities beslistabelgebaseerde systemen (behorend bij hoofdstuk 2, paragraaf 2.1.2)

De Declaration Set bestaat uit een set van goal literals, een set van input literals en een set van semantische contradicties (constraints).

1. **Declaration Set (D)** = $\mathbf{G} \cup \mathbf{L} \cup \mathbf{C}$
   $\mathbf{G} = \{G_1, \ldots, G_p\}$ is de set goal literals (alle mogelijke output of conclusies)
   $\mathbf{I} = \{I_1, \ldots, I_p\}$ is de set input literals (alle mogelijke input)
   $\mathbf{C} = \{C_1, \ldots, C_r\}$ is de set semantische contradicties: $C_i = \{L_{i1}, \ldots, L_{im}\}$
   met $L_{i1} \wedge \ldots \wedge L_{im}$ is een semantische contradictie

Een omgeving (Environment) is een subset van alle input literals I die geen semantische contradictie vormen:

2. **Environment (E)**: $\neg(\mathbf{E} \rightarrow C\sigma)$ voor alle $C \in \mathbf{C}$ en voor elke substitutie $\sigma$ waarbij $C\sigma$ staat voor een instantie van C die wordt gevormd via een substitutie $\sigma$ in C.

De set van alle hypotheses is de set van alle actiewaarden van alle tabellen:

3. **Hypothese (H)**: $H \in \mathbf{H}$ iff $(\exists DT_n \in \mathbf{DT})$ $(H \in \mathbf{AV}_n)$

De set van afleidbare hypotheses is de set van actiewaarden die afgeleid kunnen worden vanuit een mogelijke environment E:

4. **Afleidbare hypothese (A)**: $A \in \mathbf{A}$ iff $(\exists E(\text{infer}(A, \mathbf{DT}, E))$
   met: infer(A, **DT**, E) iff $(\mathbf{DT} \cup E) \vdash A$

# APPENDIX B
## Voorbeelden van anomalieën

In deze appendix zullen voorbeelden worden gegeven van de verschillende categorieën van anomalieën zoals besproken in hoofdstuk 2, paragraaf 2. Bepaalde anomalieën kunnen alleen worden vastgesteld aan de hand van een Input Set, Goal Set en een set van semantische contradicties. Het metamodel in Figuur 1* heeft betrekking op alle voorbeelden in dit hoofdstuk.

| Attributen | Type | Multivalued | Domein |
|---|---|---|---|
| wijnkleur | string | nee | {rood, wit, rose} |
| wijnsmaak | string | ja | {krachtig, soepel, vol, droog, zoet} |
| wijnprijs | integer | nee | {0 ... 1000} |
| wijnstreek | string | nee | {Bordeaux, Bourgogne, ... , Riesling} |
| wijnmerk | string | nee | {MerkA, MerkB, .... , MerkF} |

| Input Set | {wijnkleur, wijnsmaak} |
|---|---|

| Goal Set | {wijnmerk, wijnprijs} |
|---|---|

| Semantische Contradicties |
|---|
| NIET ((wijnkleur = 'rood') EN (wijnsmaak = 'zoet')) |
| NIET ((wijnkleur = 'rood') EN (wijnsmaak = 'droog')) |

**Figuur 1: metamodel**

In dit metamodel wordt aangegeven welke attributen geldig zijn en welke attribuutwaarden in het domein kunnen voorkomen. Verder wordt aangegeven wat het type van een attribuut is en of meerdere waarden zijn toegestaan (multivalued). Op deze wijze worden bepaalde simpele semantische contradicties impliciet al aangegeven, bijvoorbeeld dat wijnkleur nooit 'rood' en 'wit' tegelijk kan zijn. Daarnaast zijn een aantal specifieke semantische contradicties aangegeven in dit metamodel.

# B1 Anomalieën in regelgebaseerde systemen

## B1.1 Redundantie

ALS (wijnkleur = 'geel') DAN ... — onverenigbare conditie
ALS (wijnprijs <= -10) DAN ... — onverenigbare conditie

ALS (wijnstreek = 'Riesling') DAN ... — onverenigbare conditie
*en (wijnstreek = 'Riesling') is van geen enkele andere regel een conclusie en behoort bovendien niet tot de Input Set*

ALS (wijnkleur = 'rood') EN (wijnkleur = 'wit') DAN ... — contradictoire condities
ALS (wijnkleur = 'rood') EN (wijnsmaak = 'zoet') DAN ... — contradictoire condities

ALS ... DAN (wijnsmaa = 'droog') — onbruikbare conclusie

ALS ... DAN (wijnstreek = 'Loire') — onbruikbare conclusie
*en (wijnstreek = 'Loire') is van geen enkele andere regel een conditie en behoort bovendien niet tot de Goal Set*

ALS (wijnprijs = 14) EN (wijnkleur = 'wit') DAN (wijnstreek = 'Elzas') — subsumed regelpaar
ALS (wijnprijs = 14) DAN (wijnstreek = 'Elzas') EN (wijnmerk = 'merkC')

---

* *De wijnvoorbeelden in deze appendix zijn uit pure fantasie ontstaan. De lezer wordt geadviseerd alleen naar de logische afleidingen te kijken, en niet naar de betekenis van de regels, vooral als u toevallig een echte wijnkenner bent.*

ALS (wijnprijs >= 11) DAN (wijnmerk = 'Bourgogne') — overbodig regelpaar
ALS (wijnprijs < 11) DAN (wijnmerk = ' Bourgogne ')

ALS (wijnkleur = 'rood') DAN (wijnmerk = 'merkA') — subsumed regel door keten

ALS (wijnkleur = 'rood') DAN (wijnsmaak = 'krachtig')
ALS (wijnsmaak = 'krachtig') DAN (wijnstreek = 'Bordeaux')
ALS (wijnstreek = 'Bordeaux') DAN (wijnmerk = 'merkA') EN (wijnprijs = 5)

## B1.2 Contradictie

ALS ... DAN (wijnkleur = 'rood') EN NIET(wijnkleur = 'rood') — contradictoire conclusies
ALS ... DAN (wijnprijs > 6) EN (wijnprijs < 5) — contradictoire conclusies

ALS (wijnkleur = 'rood') DAN (wijnsmaak = 'zoet') — contradictoire regel

ALS (wijnsmaak = 'vol') DAN (wijnstreek = 'Bourgogne') — contradictoir regelpaar
ALS (wijnstreek = 'Bourgogne') DAN NIET(wijnsmaak = 'vol')

ALS (wijnsmaak = 'soepel') DAN (wijnstreek = 'Bourgogne') — contradictoir regelpaar
ALS (wijnsmaak = 'soepel') DAN (wijnstreek = 'Bordeaux')

ALS (wijnprijs = 7) DAN (wijnkleur = 'rose') — contradictoire keten
ALS (wijnprijs = 7) DAN (wijnsmaak = 'droog')
ALS (wijnsmaak = 'droog') DAN (wijnkleur = 'wit')

## B1.3 Circulariteit

ALS (wijnkleur = 'wit') EN ... DAN (wijnkleur = 'wit') — circulaire regel

ALS (wijnprijs > 5) DAN (wijnsmaak = 'krachtig') — circulair regelpaar
ALS (wijnsmaak = 'krachtig') DAN (wijnprijs = 6)

ALS (wijnprijs = 7) DAN (wijnkleur = 'rose') — circulaire keten
ALS (wijnprijs = 7) DAN (wijnsmaak = 'droog')
ALS (wijnsmaak = 'droog') DAN (wijnmerk = 'merkE')
ALS (wijnmerk = 'merkE') DAN (wijnprijs >= 6)

## B1.4 Onvolledigheid

ALS (wijnkleur = 'wit') DAN ... — ongebruikte input
ALS (wijnkleur = 'rose') DAN ...
*en de Business Logica bevat geen regel: ALS (wijnkleur = 'rood') DAN ...*

ALS (wijnprijs < 10) DAN ... — ongebruikte input
ALS (wijnprijs > 15) DAN ...
*en de Business Logica bevat geen regels die de rest van het domein van wijnprijs afdekken*

Onbruikbare output:
*Stel de Business Logica bevat alleen de volgende regels:*

ALS (wijnkleur = 'rood') EN (wijnsmaak = 'vol') DAN (wijnprijs = 7) EN (wijnmerk = 'merkA')
ALS (wijnkleur = 'rood') EN (wijnsmaak = 'krachtig') DAN (wijnprijs = 8) EN (wijnmerk = 'merkB')
ALS (wijnkleur = 'rose') DAN (wijnprijs = 6) EN (wijnmerk = 'merkC')
ALS (wijnkleur = 'wit') DAN (wijnstreek = 'Riesling')

*Er is nu onbruikbare output, aangezien geen van de goalattributen (wijnprijs en wijnmerk) worden afgeleid in de situatie wijnkleur = 'wit'*

Ontbrekende output:
*Zie het voorbeeld bij onbruikbare output, maar dan met één regel extra:*

ALS (wijnkleur = 'wit') DAN (wijnmerk = 'MerkF')

*Er is nu ontbrekende output aangezien niet alle goalattributen kunnen worden afgeleid in de situatie wijnkleur = 'wit'*

# B2 Anomalieën in tabelgebaseerde systemen

De anomalieën die voorkomen in systemen gebaseerd op regels, kunnen tevens voorkomen in systemen gebaseerd op beslistabellen. Op basis van de voorbeelden die zijn gegeven voor regelgebaseerde systemen, kan men gemakkelijk een voorstelling maken van hoe deze er uit kunnen zien in tabelgebaseerde systemen. Er zal daarom slechts een beperkt aantal anomaliecategorieën worden getoond in de volgende voorbeelden. Er zullen voorbeelden worden gegeven van zowel intra-tabulaire anomalieën (anomalieën binnen één tabel) als inter-tabulaire anomalieën (anomalieën die plaatsvinden binnen meerdere tabellen).

## B2.1 Redundantie (intra-tabulair):

| **DT 1** | | 1 | 2 | 3 |
|---|---|---|---|---|
| C1 | wijnkleur | wit | wit | * |
| C2 | wijnprijs | > 10 | * | < 0 |
| A1 | wijnstreek | Riesling | Riesling | Elzas |
| A2 | wijnmerk | MerkC | MerkC | MerkF |

**In DT 1:**
subsumed kolompaar: *kolom 2 maakt kolom 1 overbodig*
onverenigbare conditie: *kolom 3: de conditie C2 (wijnprijs < 0) is een conditie waaraan nooit voldaan kan worden*

| **DT 2** | | 1 | 2 | 3 |
|---|---|---|---|---|
| C1 | wijnkleur | rood | wit | rose |
| C2 | wijnmerk | * | * | * |
| A1 | wijnprijs | 6 | 5 | 5 |
| A2 | wijnprijs | 6 | 5 | * |

**In DT 2:**
redundante conditie rij: *rij C2: irrelevante conditie voor deze tabel*
redundante actie rij: *rij A2: tweemaal hetzelfde actie onderwerp; A2 is overbodig*

## B2.2 Contradictie (intra-tabulair):

| **DT 3** | | 1 | 2 | 3 |
|---|---|---|---|---|
| C1 | wijnkleur | wit | wit | rose |
| C2 | wijnprijs | > 10 | > 12 | < 10 |
| A1 | wijnstreek | Riesling | Elzas | * |
| A2 | wijnmerk | * | * | MerkC |
| A3 | wijnmerk | * | * | MerkD |

**In DT 3:**
contradictoir kolompaar: *kolom 1 en 2: condities kolom 2 zijn een subset van de condities van kolom 1, echter de acties zijn contradictoir*

contradictoire conclusies: *rij A2 en A3: tweemaal hetzelfde actierij onderwerp maar met conflicterende actiewaarden bij dezelfde conditiewaarden (kolom 3)*

## B2.3 Circulariteit (intra-tabulair):

| DT 4 | | 1 | 2 |
|---|---|---|---|
| C1 | wijnkleur | rood | rood |
| C2 | wijnprijs | < 7 | > 9 |
| A1 | wijnstreek | Bordeaux | Bourgogne |
| A2 | wijnkleur | rood | rood |

**In DT 4:**
circulariteit: *conditie C1 hangt af van de waarde van actie A2*

## B2.4 Onvolledigheid (intra-tabulair):

**In DT 4:**
ongebruikte input: *conditiewaarden van zowel C1 en C2 omvatten niet het volledige domein.*

## B2.5 Redundantie (inter-tabulair):

| DT 5 | | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| C1 | wijnkleur | rood | rood | wit | wit | rose | rose |
| C2 | wijnsmaak | krachtig | vol | droog | zoet | droog | zoet |
| A1 | wijnprijs | 5 | 6 | 5 | 4 | 5 | 4 |
| A2 | wijnstreek | Bordeaux | Bourgogne | Riesling | Riesling | Riesling | Riesling |

| DT 6 | | 1 | 2 |
|---|---|---|---|
| C1 | wijnsmaak | droog | zoet |
| A1 | wijnprijs | 5 | 4 |
| A2 | wijnmerk | MerkC | MerkD |

| DT 7 | | 1 | 2 |
|---|---|---|---|
| C1 | wijnkleur | wit | rose |
| C2 | ... | ... | ... |
| A1 | wijnstreek | Riesling | Riesling |

**In DT 5, 6 en 7:**
redundante kolommen door keten: *de kolommen 3, 4, 5 en 6 uit DT 5 zijn overbodig: deze worden al in DT 6 en DT 7 beschreven*

| DT 8 | | 1 | 2 | 3 |
|---|---|---|---|---|
| C1 | ^wijnprijs | < 5 | 5 | > 5 |
| A1 | ... | ... | ... | ... |
| A2 | ... | ... | ... | ... |

| DT 9 | | 1 | 2 | 3 |
|---|---|---|---|---|
| C1 | wijnkleur | rood | wit | [ ] |
| A1 | wijnprijs | 5 | 6 | 5 |

| DT 10 | | 1 | 2 |
|---|---|---|---|
| C1 | wijnprijs | > 8 | [ ] |
| A1 | wijnmerk | MerkA | ... |

| DT 11 | | 1 | 2 |
|---|---|---|---|
| C1 | wijnprijs | > 8 | [ ] |
| C2 | wijnstreek | Bordeaux | ... |
| A1 | wijnmerk | MerkA | ... |

**In DT 8, 9, 10 en 11:**
onverenigbare condities: *de condities van kolom 1 en 2 van DT 8 zijn onverenigbaar aangezien in DT 9 aangegeven wordt dat wijnprijs nooit aan deze condities zal voldoen*

subsumed kolompaar: *kolom 1 van DT 10 maakt kolom 1 van DT 11 overbodig*

## B2.6 Contradictie (inter-tabulair):

| DT 12 | | 1 | 2 |
|---|---|---|---|
| C1 | wijnprijs | > 12 | ... |
| C2 | wijnstreek | * | ... |
| A1 | wijnmerk | MerkF | ... |

| DT 13 | | 1 | 2 |
|---|---|---|---|
| C1 | wijnprijs | > 11 | ... |
| C2 | wijnkleur | * | ... |
| A1 | wijnmerk | MerkE | ... |

**In DT 12 en 13:**
contradictoire conclusies: *kolom 1 van DT 12 en kolom 1 van DT 13 bevatten contradictoire conclusies.*

## B2.7 Circulariteit (inter-tabulair):

| DT 14 | | 1 | 2 |
|---|---|---|---|
| C1 | ^wijnstreek | Riesling | ... |
| C2 | wijnsmaak | droog | ... |
| A1 | wijnmerk | MerkD | ... |

| DT 15 | | 1 | 2 |
|---|---|---|---|
| C1 | ^wijnmerk | MerkD | ... |
| C2 | wijnkleur | * | ... |
| A1 | wijnstreek | Riesling | ... |

**In DT 14 en 15:**
circulariteit: *conditie C1 van DT 14 is afhankelijk van A1 van DT 15, welke alleen afgeleid kan worden middels A1 van DT 14*

# B3 Anomalieën in combinaties van regels en tabellen

## B3.1 Redundantie:

Subsumed regel-kolompaar:

ALS (wijnkleur = 'rood') EN (wijnsmaak = 'krachtig') DAN (wijnstreek = 'Bordeaux')

| DT 16 | | 1 | 2 |
|---|---|---|---|
| C1 | wijnkleur | rood | ... |
| C2 | wijnsmaak | * | ... |
| A1 | wijnstreek | Bordeaux | ... |

## B3.2 Contradictie:

Contradictoire keten:

ALS (wijnkleur = 'wit') DAN (wijnstreek = 'Riesling')
ALS (wijnsmaak = 'droog') DAN (wijnstreek = 'Champagne')

| DT 17 | | 1 | 2 |
|---|---|---|---|
| C1 | wijnkleur | wit | ... |
| C2 | wijnstreek | Riesling | ... |
| A1 | wijnsmaak | droog | ... |

## B3.3 Circulariteit:

Circulair combinatiepaar:

ALS (wijnsmaak = 'droog') DAN (wijnkleur = 'wit')

| DT 18 | | 1 | 2 |
|---|---|---|---|
| C1 | wijnkleur | wit | ... |
| C2 | wijnstreek | * | ... |
| A1 | wijnsmaak | droog | ... |

# APPENDIX C
## Complete beschrijving van de algoritmen

De algoritmen in deze appendix zijn beschreven in pseudo-code. De volgende symbolen met bijbehorende betekenis worden gebruikt:

| | |
|---|---|
| A == B | A is gelijk aan B |
| A != B | A is ongelijk aan B |
| A = B | A wordt B |

# C1 Algemeen bruikbare algoritmen

**Literal A gelijk aan Literal B:**

```
ALS LiteralA.Attribuut == LiteralB.Attribuut EN LiteralA.Waarde == LiteralB.Waarde DAN
   ALS LiteralA.Relatie == LiteralB.Relatie EN LiteralA.Valentie == LiteralB.Valentie DAN
         IsGelijkAan (LiteralA, LiteralB)= WAAR
   ALS LiteralA.Relatie == '<' EN LiteralB.Relatie == '>=' EN LiteralA.Valentie != LiteralB.Valentie DAN
         IsGelijkAan (LiteralA, LiteralB) = WAAR
   ALS LiteralA.Relatie == '>' EN LiteralB.Relatie == '<=' EN LiteralA.Valentie != LiteralB.Valentie DAN
         IsGelijkAan (LiteralA, LiteralB) = WAAR
   ALS LiteralA.Relatie == '<=' EN LiteralB.Relatie == '>' EN LiteralA.Valentie != LiteralB.Valentie DAN
         IsGelijkAan (LiteralA, LiteralB) = WAAR
   ALS LiteralA.Relatie == '>=' EN LiteralB.Relatie == '<' EN LiteralA.Valentie != LiteralB.Valentie DAN
         IsGelijkAan (LiteralA, LiteralB) = WAAR
```

**Literal A negatie van Literal B:**

```
ALS LiteralA.Attribuut == LiteralB.Attribuut EN LiteralA.Waarde == LiteralB.Waarde DAN
   ALS LiteralA.Relatie == LiteralB.Relatie EN LiteralA.Valentie != LiteralB.Valentie DAN
         IsNegatieVan (LiteralA, LiteralB) = WAAR
   ALS LiteralA.Relatie == '<' EN LiteralB.Relatie == '>=' EN LiteralA.Valentie == LiteralB.Valentie DAN
         IsNegatieVan (LiteralA, LiteralB) = WAAR
   ALS LiteralA.Relatie == '>' EN LiteralB.Relatie == '<=' EN LiteralA.Valentie == LiteralB.Valentie DAN
         IsNegatieVan (LiteralA, LiteralB) = WAAR
         enz
         enz
```

**Literal is element van Input Set:**

```
ALS Literal van type string DAN
   VOOR ALLE InputSetAttributen DOE
      ALS Literal.Attribuut == InputSetAttribuut.Naam DAN
         VOOR ALLE InputSetAttribuut.Domeinwaarden DOE
            ALS Literal.Waarde == InputSetAttribuut.Domeinwaarde DAN IsElementInputSet = WAAR

ALS Literal van type integer DAN
   ALS Literal.Attribuut ELEMENT VAN InputAttributen DAN
      ALS Literal.Waarde <= InputSetAttribuut.Maximumwaarde EN
           Literal.Waarde >= InputSetAttribuut.Minimumwaarde DAN IsElementInputSet = WAAR
```

**Literal is element van Goal Set:**

Analoog aan **Literal is element van Input Set**, maar dan niet met InputSetAttributen maar met GoalSetAttributen

**Literal A afleidbaar uit Literal B:**

```
ALS LiteralA van type string en LiteralB van type string DAN
   ALS LiteralA.Valentie == "!" DAN
      ALS LiteralA.Attribuut ELEMENT VAN AttributenSet DAN
         ALS LiteralA.Attribuut IS SINGLEVALUED
            ALS LiteralA.Attribuut == LiteralB.Attribuut EN
                  LiteralA.Waarde != LiteralB.Waarde EN
                  LiteralA.Negatie != LiteralB.Negatie DAN IsAfleidbaarUit (LiteralA, LiteralB)= WAAR

ALS LiteralA van type integer en LiteralB van type integer DAN
   ALS LiteralA.Attribuut == LiteralB.Attribuut DAN
      ALS LiteralA.Valentie == LiteralB.Valentie DAN
         ALS LiteralA.Relatie == '<' EN LiteralB.Relatie == '<' EN
               LiteralA.Waarde >= LiteralB.Waarde DAN IsAfleidbaarUit (LiteralA, LiteralB)= WAAR
         ALS LiteralA.Relatie == '<' EN LiteralB.Relatie == '<=' EN
               LiteralA.Waarde > LiteralB.Waarde DAN IsAfleidbaarUit (LiteralA, LiteralB)= WAAR
          enz
          enz
```

**Literal A is onverenigbaar met Literal B:**

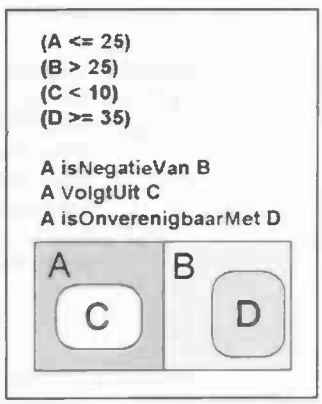
```
ALS LiteralA van type integer en LiteralB van type integer DAN
   ALS LiteralA.Attribuut == LiteralB.Attribuut DAN
      ALS LiteralA.Valentie == LiteralB.Valentie DAN
         ALS LiteralA.Relatie == '<' EN LiteralB.Relatie == '>' EN
               LiteralA.Waarde <= LiteralB.Waarde DAN IsOnverenigbaarMet (LiteralA, LiteralB)= WAAR
         ALS LiteralA.Relatie == '<' EN LiteralB.Relatie == '>=' EN
               LiteralA.Waarde < LiteralB.Waarde DAN IsOnverenigbaarMet (LiteralA, LiteralB)= WAAR
          enz
          enz
```

**Verzameling van Literals bevat syntactische contradictie:**

```
VOOR ALLE Literals i DOE
   VOOR ALLE Literals j DOE
      ALS IsNegatieVan(Literal i, Literal j) DAN
                  OUTPUT "syntactische contradictie, A is negatie van B"
      ALS IsOnverenigbaarMet(Literal i, Literal j) DAN
                  OUTPUT "syntactische contradictie, A is onverenigbaar met B"
```

**Verzameling van Literals bevat semantische contradictie:**

```
VOOR ALLE Literals i DOE
   VOOR ALLE Literals j DOE
      ALS Literal i.Attribuut == Literal j.Attribuut EN
            Literal i.Waarde != Literal j.Waarde EN
            Literal i.Relatie == Literal j. Relatie EN
            Literal i.Attribuut == SINGLEVAL DAN
                  OUTPUT "semantische contradictie, attribuut is niet multivalued"
VOOR ALLE Semantische contradicties s DOE
   SemConGevonden = WAAR
   VOOR ALLE SemConLiterals sl DOE
      LitGevonden = ONWAAR
      VOOR ALLE Literals DOE
         ALS isGelijkAan (SemConLiteral sl, Literal) DAN LitGevonden = WAAR
         ALS isAfleidbaarUit (SemConLiteral sl, Literal) DAN LitGevonden = WAAR
      ALS LitGevonden == ONWAAR DAN SemConGevonden = ONWAAR
   ALS SemConGevonden == WAAR DAN
                  OUTPUT "semantische contradictie s gevonden"
```




**Verzameling A van Literals vormt subset van Verzameling B van Literals:**

```
SubsetAB = WAAR
VOOR ALLE Literals van A DOE
   LiteralGevonden = ONWAAR
   VOOR ALLE Literals van B DOE
      ALS isGelijkAan(Literal van A, Literal van B) DAN LiteralGevonden = WAAR
      ALS isAfleidbaarUit(Literal van A, Literal van B) DAN LiteralGevonden = WAAR
      ALS LiteralGevonden == ONWAAR DAN SubsetAB ONWAAR
```

# C2 Algoritmen voor integriteittests

**Redundantie, Onverenigbare Condities:**

```
VOOR ALLE Regels r DOE
  VOOR ALLE Condities ra DOE
    onverenigbaar = WAAR
    ALS IsElementInputSet(Conditie ra) DAN onverenigbaar = ONWAAR
    VOOR ALLE Regels r DOE
      VOOR ALLE Conclusies rc DOE
        ALS IsGelijkAan (Conditie ra, Conclusie rc) DAN onverenigbaar = ONWAAR
        ALS IsAfleidBaarUit (Conditie ra, Conclusie rc) DAN onverenigbaar = ONWAAR
    ALS onverenigbaar == WAAR DAN OUTPUT "onverenigbare conditie ra in regel r"
```

**Redundantie, Onbruikbare Conclusies:**

```
VOOR ALLE Regels r DOE
  VOOR ALLE Conclusies rc DOE
    onbruikbaar = WAAR
    ALS IsElementGoalSet(Conclusie rc) DAN onbruikbaar = ONWAAR
    VOOR ALLE Regels r DOE
      VOOR ALLE Condities ra DOE
        ALS IsGelijkAan (Conclusie rc, Conditie ra) DAN onbruikbaar = ONWAAR
        ALS IsAfleidBaarUit (Conditie ra, Conclusie rc) DAN onbruikbaar = ONWAAR
    ALS onbruikbaar == WAAR DAN OUTPUT "onbruikbare conclusie rc in regel r"
```

**Redundantie, Contradictoire Condities:**

```
VOOR ALLE Regels r DOE
  ALS bevatContradictie(Condities van r) DAN OUTPUT "contradictoire condities in regel r"
```

**Contradictie, Contradictoire Conclusies:**

```
VOOR ALLE Regels r DOE
  ALS bevatContradictie(Conclusies van r) DAN OUTPUT "contradictoire conclusies in regel r"
```

**Contradictie, Contradictoire Regel:**

```
VOOR ALLE Regels r DOE
  ALS NIET bevatContradictie(Condities van r) EN
       NIET bevatContradictie(Conclusies van r) EN
       bevatContradictie(Condities van r PLUS Conclusies van r) DAN
        OUTPUT "contradictoire regel: regel r"
```

**Circulariteit, Circulaire Regel:**

```
VOOR ALLE Regels r DOE
  VOOR ALLE Condities ra DOE
    VOOR ALLE Conclusies rc DOE
      ALS Conditie ra.Attribuut == Conclusies rc.Attribuut DAN OUTPUT "circulaire regel: regel r"
```

# C3 Algoritmen voor regeltests

**Redundantie, subsumed of dubbel regelpaar:**

```
VOOR ALLE Regels r DOE
  VOOR ALLE ANDERE Regels p DOE
    ALS Subset(Condities p, Condities r)
      ALS Subset(Conclusies r, Conclusies p)
        ALS Conclusies r.Aantal == Conclusies p.Aantal EN
            Condities r.Aantal == Condities p.Aantal DAN
              OUTPUT "redundantie: dubbel regelpaar: regel p en regel r"
        ANDERS DAN
              OUTPUT "redundantie: regel p subsumes regel r"
```

**Redundantie, contradictoir regelpaar:**

```
VOOR ALLE Regels r DOE
  VOOR ALLE ANDERE Regels p DOE
    ALS (NIET BevatContradictie(Conclusies r PLUS Condities r) EN
         NIET BevatContradictie(Conclusies p PLUS Condities p) DAN
      ALS Subset(Condities p, Condities r)
        ALS BevatContradicties(Condities p PLUS Conclusies r PLUS Conclusies p) DAN
              OUTPUT "contradictoir regelpaar: regel r en regel p"
      ALS Subset(Condities p, Conclusies r)
        ALS BevatContradicties(Condities r PLUS Conclusies r PLUS Conclusies p) DAN
              OUTPUT "contradictoir regelpaar: regel r en regel p"
```

**Redundantie, overbodig regelpaar:**

```
VOOR ALLE Regels r DOE
  VOOR ALLE ANDERE Regels p DOE
    ALS Condities r.Aantal == Condities p.Aantal == 1 DAN
      ALS Conditie r NEGATIE VAN Conditie p DAN
        ALS Conclusies r.Aantal == Conclusies p.Aantal DAN
          ALS Subset (Conclusies r, Conclusies p) DAN
              OUTPUT "overbodig regelpaar (contradictoire condities): regel r en regel p"
```

**Circulariteit, circulair regelpaar:**

```
VOOR ALLE Regels r DOE
  VOOR ALLE ANDERE Regels p DOE
    VOOR ALLE Condities van r DOE
      VOOR ALLE Conclusies van p DOE
        ALS Conditie r.Attribuut == Conclusie p.Attribuut DAN
          VOOR ALLE Conclusies van r DOE
            VOOR ALLE Condities van p DOE
              ALS Conclusie r.Attribuut == Conditie p.Attribuut DAN
              OUTPUT "circulair regelpaar: regel r en regel p"
```

# C4 Algoritmen voor regelextensietests

**Creëren van mogelijke environments**

```
VOOR ALLE InputAttributen DOE
   ALS InputAttribuut van type string DAN

      ALS InputAttribuut != multivalued DAN
         VOOR ALLE Environments e
            VOOR ALLE Domeinwaarden w van InputAttribuut DOE
               MAAK NIEUW Environment n
               VOEG Environment e TOE AAN Environment n
               ALS NIET BevatContradictie(Environment n PLUS Literal(InputAttribuut '=' Waarde w))DAN
                  VOEG Literal(InputAttribuut '=' Waarde w) TOE AAN Environment n
         VOEG ALLE NIEUWE Environments n TOE aan Environments
      ALS InputAttribuut a == multivalued DAN
         VOOR ALLE Environments e
            VOOR ALLE Domeinwaarden w van InputAttribuut a DOE
               ALS NIET BevatContradictie(Environment n PLUS Literal(InputAttribuut '=' Waarde w))DAN
                  VOEG Literal(InputAttribuut a = Waarde w) TOE AAN Environment e
RETURN Environments
```

**Creëren van een regelextensie op basis van Environment e**

```
AfgeleideConclusies = nul
GebruikteCondities = nul
GevuurdeRegels = nul
NietsVuurbaarMeer = ONWAAR
HERHAAL TOT NietsVuurbaarMeer = WAAR
   IetsGevuurd = ONWAAR
   VOOR ALLE Regels i DOE
      ALS Regel i GEEN ELEMENT VAN GevuurdeRegels DAN
         Vuurbaar = WAAR
         VOOR ALLE Regelcondities j DOE
            ALS Regelconditie j GEEN ELEMENT VAN (Environment e OR AfgeleideConclusies)
               DAN Vuurbaar = ONWAAR
         ALS Vuurbaar = WAAR DAN
            Afgeleide Conclusies = AfgeleideConclusies + Regelconclusie i
            GevuurdeRegels = GevuurdeRegels + Regel i
            IetsGevuurd = WAAR
   ALS IetsGevuurd = WAAR DAN NietsVuurbaarMeer = ONWAAR
   ALS IetsGevuurd = ONWAAR DAN NietsVuurbaarMeer = WAAR
RETURN GevuurdeRegels
```

**Contradictie: contradictoire keten**

```
VOOR ALLE MogelijkeEnvironments e DOE
   VOOR ALLE GevuurdeRegels r van de regelextensie van e DOE
      VOOR ALLE GevuurdeRegels p > r+2 DOE
         ALS NIET BevatContradictie(antec van r PLUS concl van r) EN
               NIET BevatContradictie(concl van p) DAN
            ALS BevatContradictie(antec van r PLUS concl van r PLUS concl van p) DAN
                  OUTPUT "contradictoire keten in environment e: Regel a, Regel b ...Regel n"
```

**Circulariteit: circulaire keten**

```
VOOR ALLE MogelijkeEnvironments e DOE
   VOOR ALLE GevuurdeRegels r van de regelextensie van e DOE
      VOOR ALLE Condities van r DOE
         VOOR ALLE GevuurdeRegels p  > r+2 DOE
            VOOR ALLE Conclusies van p DOE
               ALS Conditie r.Attribuut == Conclusie p.Attribuut DAN
                     OUTPUT: "circulaire keten in environment e: Regel a, Regel b ...Regel n"
```

**Redundantie: subsumed regel door keten**

```
VOOR ALLE Regels r DOE
  ALS Regel r GEEN DEEL VAN OUTPUT van Test_OnverenigbareCondities EN
              GEEN DEEL VAN OUTPUT van Test_ContradictoireCondities EN
              GEEN DEEL VAN OUTPUT van Test_OnbruikbareConclusies EN
              GEEN DEEL VAN OUTPUT van Test_SubsumedRegelpaar EN
              GEEN DEEL VAN OUTPUT van Test_OverbodigRegelpaar DAN
    RedundanteRegel = WAAR
    VOOR ALLE MogelijkeEnvironments e DOE

      ALS NIET isSubset(Afgeleide Conclusies MET r , Afgeleide Conclusies ZONDER r DAN
        RedundanteRegel = ONWAAR
    ALS RedundanteRegel = WAAR DAN
      OUTPUT: "subsumed regel door keten: alle conclusies van regel r worden ook elders afgeleid,
        in ieder mogelijk environment"
```

# C5 Algoritmen voor onvolledigheidtests

**Ongebruikte input**

```
VOOR ALLE InputAttributen DOE
  AttribuutGebruikt = ONWAAR
  VOOR ALLE GoalAttributen DOE
    ALS GoalAttribuut == InputAttribuut DAN AttribuutGebruikt = WAAR
  ALS AttribuutGebruikt == ONWAAR DAN

  MAAK InputLiterals voor InputAttribuut
  VOOR ALLE InputLiterals t DOE
    LitGebruikt = ONWAAR
    VOOR ALLE Regels r DOE
      VOOR ALLE Regelcondities c DOE
        ALS IsGelijkAan(Regelconditie c, InputLiteral t) DAN LitGebruikt = WAAR
        ALS IsAfleidbaarUit(Regelcondite c, InputLiteral t) DAN LitGebruikt = WAAR
    ALS LitGebruikt == ONWAAR DAN
      OUTPUT "ongebruikte input: InputLiteral t wordt in geen enkele regel gebruikt"
```

**Onbruikbare Output**

```
EenGoalAfgeleid = ONWAAR
VOOR ALLE GoalAttributen g DOE
  ALS GoalAttribuut g ELEMENT VAN InputAttributen DAN EenGoalAfgeleid = WAAR
VOOR ALLE MogelijkeEnvironments e DOE
  VOOR ALLE GevuurdeRegels r van de regelextensie van e DOE
    VOOR ALLE Conclusies van r DOE
      VOOR ALLE GoalAttributen g DOE
        ALS Conclusie.Attribuut == GoalAttribuut g DAN EenGoalAfgeleid = WAAR
  ALS EenGoalAfgeleid == ONWAAR DAN
      OUTPUT "ontbruikbare output: geen enkel goalattribuut wordt afgeleid in environment e"
```

**Ontbrekende Output**

```
VOOR ALLE MogelijkeEnvironments e DOE
  VOOR ALLE Gewenste GoalAttributen g behorend bij e DOE
    GoalAttribAfgeleid = ONWAAR
    ALS GoalAttribuut g ELEMENT VAN InputAttributen DAN GoalAttribAfgeleid = WAAR
    VOOR ALLE GevuurdeRegels r van de regelextensie van e DOE
      VOOR ALLE Conclusies van r DOE
        ALS Conclusie.Attribuut == Gewenste GoalAttribuut g DAN GoalAttribAfgeleid = WAAR
    ALS GoalAttribAfgeleid == ONWAAR DAN
      OUTPUT "ontbrekende output: Gewenst goalattribuut g wordt niet afgeleid in environment e"
```

# APPENDIX D
## Test Knowledge Bases

## KB-A

```
--- Deze Knowledge Base bevat voorbeelden van alle simpele anomalieen,
--- dat wil zeggen, die met integriteittests en regeltests kunnen worden
--- gedetecteerd.
--- Er zijn echter ook veel complexe anomalieen die mede ontstaan vanwege
--- deze simpele anomalieen. Het is daarom overzichtelijker om alleen de
--- integriteittests (binnen 1 regel) en de regeltests (binnen een paar)
--- uit te voeren op deze KB.


ATTRIBUUT_STR wijnkleur  (rood, wit) singlevalued
ATTRIBUUT_STR wijnsmaak  (vol, krachtig, soepel, droog, zoet, fruitig) singlevalued
ATTRIBUUT_STR wijnstreek  (Bordeaux, Bourgogne, Elzas, Loire) singlevalued
ATTRIBUUT_INT wijnprijs  (0, 400)
ATTRIBUUT_INT percentage  (0, 20)

INPUT ATTRIBUTEN wijnkleur, wijnsmaak, wijnstreek, wijnprijs
GOAL  ATTRIBUTEN wijnkleur, wijnsmaak, wijnstreek, wijnprijs, percentage

SEMCON (wijnkleur='rood') EN (wijnsmaak='zoet')
SEMCON (wijnstreek='Elzas') EN (percentage=11)

--- onverenigbare condities:
ALS (wijnkleur='geel') DAN (wijnsmaak='vol')
ALS (wijnprijs>=500) DAN (wijnsmaak='vol')

--- contradictoire condities:
ALS (wijnkleur='rood') EN !(wijnkleur='rood') DAN (wijnsmaak='vol')
ALS (wijnkleur='rood') EN (wijnsmaak='zoet') DAN (wijnprijs=14)

--- onbruikbare conclusies:
ALS (wijnkleur='rood') DAN (wijnsmak='soepel')

--- contradictoire conclusies:
ALS (wijnkleur='wit') EN (wijnstreek='Bordeaux') DAN (wijnsmaak='droog') EN !(wijnsmaak='droog')
ALS (wijnkleur='wit') EN (wijnstreek='Bordeaux') DAN (wijnprijs>=10) EN (wijnprijs<8)

--- contradictoire regels
ALS (wijnstreek='Loire') DAN !(wijnstreek='Loire')
ALS (wijnkleur='rood') DAN (wijnsmaak='zoet')

--- circulaire regels
ALS (wijnprijs=9) DAN (wijnprijs=9)

--- subsumed regelpaar:
ALS (wijnsmaak='krachtig') EN (wijnkleur='wit') DAN (percentage>12)
ALS (wijnsmaak='krachtig') DAN (percentage>13)

--- dubbel regelpaar:
ALS (wijnsmaak='zoet') DAN (wijnprijs>5)
ALS (wijnsmaak='zoet') DAN !(wijnprijs<=5)

--- contradictoir regelpaar:
ALS (wijnstreek='Elzas') DAN (wijnprijs=7)
ALS (wijnprijs<=7) DAN (percentage=11)
ALS (wijnstreek='Bourgogne') EN (wijnsmaak='soepel') DAN (wijnprijs=13)
ALS (wijnstreek='Bourgogne') DAN (wijnprijs=8)

--- circulair regelpaar:
ALS (wijnstreek='Elzas') DAN (wijnsmk='droog')
ALS (wijnsmk='droog') DAN (wijnstreek='Elzas')
```

# KB-B

```
--- Deze Knowledge Base bevat voorbeelden van complexere anomalieen,
--- dat wil zeggen, circulaire en contradictoire ketens, subsumed regels
--- door een keten en vormen van onvolledigheid.


ATTRIBUUT_STR systeem (langzaam, normaal, snel, super) singlevalued
ATTRIBUUT_STR geschikte_combinatie  (ja, nee) singlevalued
ATTRIBUUT_STR besturingssysteem  (win_XP, win_2000, win_98) singlevalued
ATTRIBUUT_STR gewenst_spel (AgeOfMythology, RomeTotalWar, Warcraft) singlevalued
ATTRIBUUT_INT CPU_MHz  (0, 3300)
ATTRIBUUT_INT geheugen_MB (0, 2048)
ATTRIBUUT_INT harddisk_GB (0, 200)

INPUT ATTRIBUTEN CPU_MHz, geheugen_MB, harddisk_GB, gewenst_spel
GOAL  ATTRIBUTEN geschikte_combinatie, besturingssysteem

SEMCON (CPU_MHz<800) EN (geheugen_MB<256) EN (geschikte_combinatie='ja')


ALS (CPU_MHz<800) EN (geheugen_MB<256) DAN (systeem='langzaam')
ALS (CPU_MHz<800) EN (geheugen_MB>=256) EN (geheugen_MB<768) DAN (systeem='langzaam')
ALS (CPU_MHz<800) EN (geheugen_MB>=768) DAN (systeem='langzaam')

ALS (CPU_MHz>=800) EN (CPU_MHz<2000) EN (geheugen_MB<256) DAN (systeem='normaal')
ALS (CPU_MHz>=800) EN (CPU_MHz<2000) EN (geheugen_MB>=256) EN (geheugen_MB<768) DAN
(systeem='normaal')
ALS (CPU_MHz>=800) EN (CPU_MHz<2000) EN (geheugen_MB>=768) DAN (systeem='snel')

ALS (CPU_MHz>=2000) EN (geheugen_MB<256) DAN (systeem='normaal')
ALS (CPU_MHz>=2000) EN (geheugen_MB>=256) EN (geheugen_MB<768) DAN (systeem='snel')
>>>> hieronder ontstaat onbruikbare output: geen regels voor de combinatie (CPU_MHz>=2000) EN
(CPU_MHz<2100) EN (geheugen_MB>=768)
ALS (CPU_MHz>=2100) EN (geheugen_MB>=768) DAN (systeem='super')


ALS (systeem='langzaam') DAN (besturingssysteem='win_98')
ALS (systeem='normaal') EN (geheugen_MB<256) DAN (besturingssysteem='win_2000')
ALS (systeem='normaal') EN (geheugen_MB>=256) DAN (besturingssysteem='win_XP')
ALS (systeem='snel') DAN (besturingssysteem='win_XP')
ALS (systeem='super') DAN (besturingssysteem='win_XP')


>>>> laatste regel in contradictoire keten:
ALS (besturingssysteem='win_98') EN (gewenst_spel='Warcraft') DAN (geschikte_combinatie='ja')
ALS (besturingssysteem='win_98') EN (gewenst_spel='AgeOfMythology') DAN
(geschikte_combinatie='nee')
ALS (besturingssysteem='win_98') EN (gewenst_spel='RomeTotalWar') DAN (geschikte_combinatie='nee')
>>>> hieronder ontstaat ongebruikte input:
ALS (besturingssysteem='win_2000') EN (harddisk_GB>=10) EN !(gewenst_spel='RomeTotalWar') DAN
(geschikte_combinatie='ja')
ALS (besturingssysteem='win_2000') EN (gewenst_spel='RomeTotalWar') DAN
(geschikte_combinatie='nee')
ALS (besturingssysteem='win_XP') EN (harddisk_GB>=10) EN !(gewenst_spel='RomeTotalWar') DAN
(geschikte_combinatie='ja')
ALS (besturingssysteem='win_XP') EN (gewenst_spel='RomeTotalWar') EN !(systeem='super') DAN
(geschikte_combinatie='nee')
ALS (besturingssysteem='win_XP') EN (harddisk_GB>=10) EN (gewenst_spel='RomeTotalWar') EN
(systeem='super') DAN (geschikte_combinatie='ja')

>>>> subsumed regel door keten:
ALS (systeem='normaal') EN (geheugen_MB=256) EN (gewenst_spel='RomeTotalWar') DAN
(geschikte_combinatie='nee')

>>>> laatste regel in circulaire keten:
ALS (geschikte_combinatie='ja') EN (gewenst_spel='RomeTotalWar') DAN (CPU_MHz>=2000)
```

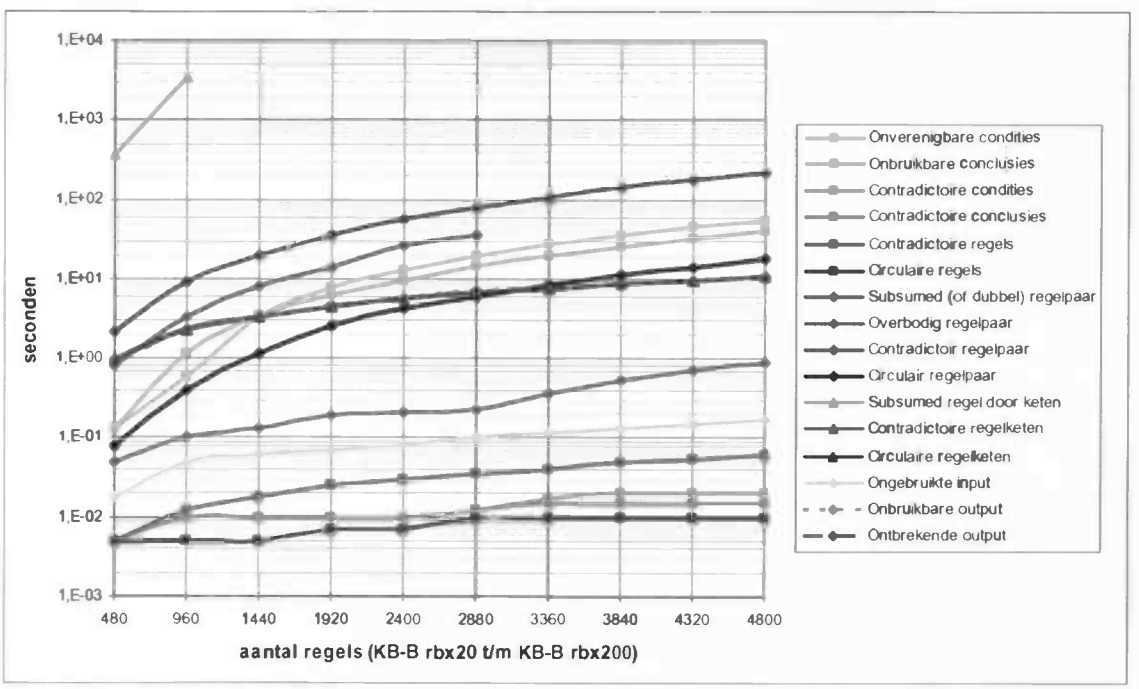
# KB-C

```
**** MOGELIJKE ATTRIBUTEN:
**** Een string-attribuut heeft de vorm:  ATTRIBUUT_STR attribuutnaam (waarde_a, waarde_b, ...)
singlevalued/multivalued
**** Een integer-attribuut heeft de vorm: ATTRIBUUT_INT attribuutnaam (minimumwaarde,
maximumwaarde)

ATTRIBUUT_STR wijnkleur  (rood, wit, rose) singlevalued
ATTRIBUUT_STR wijnsmaak  (vol, krachtig, droog, zoet, soepel, fruitig) multivalued
ATTRIBUUT_STR wijnmerk   (A,B,C,D,E,F,G,H,I) singlevalued
ATTRIBUUT_STR wijnstreek (Bordeaux, Bourgogne, Beaujolais, Moesel, Elzas, Loire) singlevalued
ATTRIBUUT_STR gerecht    (biefstuk, kip, varken, zalm, dessert, hert) singlevalued
ATTRIBUUT_INT wijnbudget (0, 1000)
ATTRIBUUT_INT wijnprijs  (0, 1000)


**** De input set en de goal set worden als volgt aangegeven:

INPUT ATTRIBUTEN wijnbudget, gerecht
GOAL  ATTRIBUTEN wijnmerk, wijnprijs


**** SEMANTISCHE CONTRADICTIES: sets van literals die nooit waar moegen zijn.

SEMCON (wijnbudget<14) EN (wijnmerk='C')


**** REGELS:
**** Een regel heeft de vorm:     ALS literal_A EN literal_B EN ... DAN literal_X EN literal_Y EN
...
**** Een literal heeft de vorm:   !(attribuutnaam = waarde) waarbij '!' betekent: NIET
**** Bij string-literals moeten de attribuutwaarden tussen '' staan.
**** Bij integer-literals zijn ook <=, >=, < en > mogelijk.


ALS (gerecht='biefstuk') DAN (wijnkleur='rood') EN (wijnsmaak='vol') EN (wijnsmaak='fruitig')
ALS (gerecht='hert')     DAN (wijnkleur='rood') EN (wijnsmaak='vol') EN (wijnsmaak='krachtig')
ALS (gerecht='varken')   DAN (wijnkleur='rose')
ALS (gerecht='kip')      DAN (wijnkleur='wit') EN (wijnsmaak='droog')
ALS (gerecht='zalm')     DAN (wijnkleur='wit') EN (wijnsmaak='vol')
ALS (gerecht='dessert')  DAN (wijnkleur='wit') EN (wijnsmaak='zoet')

ALS (wijnkleur='rood') EN (wijnsmaak='vol') EN (wijnsmaak='fruitig') DAN (wijnstreek='Beaujolais')
ALS (wijnkleur='rood') EN (wijnsmaak='vol') EN (wijnsmaak='krachtig') DAN (wijnstreek='Bordeaux')
ALS (wijnkleur='rose') DAN (wijnstreek='Bourgogne')
ALS (wijnkleur='wit') EN (wijnsmaak='droog') DAN (wijnstreek='Elzas')
ALS (wijnkleur='wit') EN (wijnsmaak='vol') DAN (wijnstreek='Loire')
ALS (wijnkleur='wit') EN (wijnsmaak='zoet') DAN (wijnstreek='Moesel')

ALS (wijnstreek='Beaujolais') EN (wijnbudget<10) DAN (wijnmerk='A')
ALS (wijnstreek='Beaujolais') EN (wijnbudget>=10) DAN (wijnmerk='B')
ALS (wijnstreek='Bordeaux') EN (wijnbudget<8) DAN (wijnmerk='A')
ALS (wijnstreek='Bordeaux') EN (wijnbudget>=8) EN (wijnbudget<12) DAN (wijnmerk='D')
ALS (wijnstreek='Bordeaux') EN (wijnbudget>=12) DAN (wijnmerk='C')
ALS (wijnstreek='Bourgogne') DAN (wijnmerk='E')
ALS (wijnstreek='Elzas') EN (wijnbudget<11) DAN (wijnmerk='F')
ALS (wijnstreek='Elzas') EN (wijnbudget>=11) DAN (wijnmerk='G')
ALS (wijnstreek='Loire') DAN (wijnmerk='H')
ALS (wijnstreek='Moesel') DAN (wijnmerk='I')

ALS (wijnmerk='A') DAN (wijnprijs=7)
ALS (wijnmerk='B') DAN (wijnprijs=10)
ALS (wijnmerk='C') DAN (wijnprijs=14)
ALS (wijnmerk='D') DAN (wijnprijs=11)
ALS (wijnmerk='E') DAN (wijnprijs=8)
ALS (wijnmerk='F') DAN (wijnprijs=9)
ALS (wijnmerk='G') DAN (wijnprijs=12)
ALS (wijnmerk='H') DAN (wijnprijs=8)
ALS (wijnmerk='I') DAN (wijnprijs=8)
```

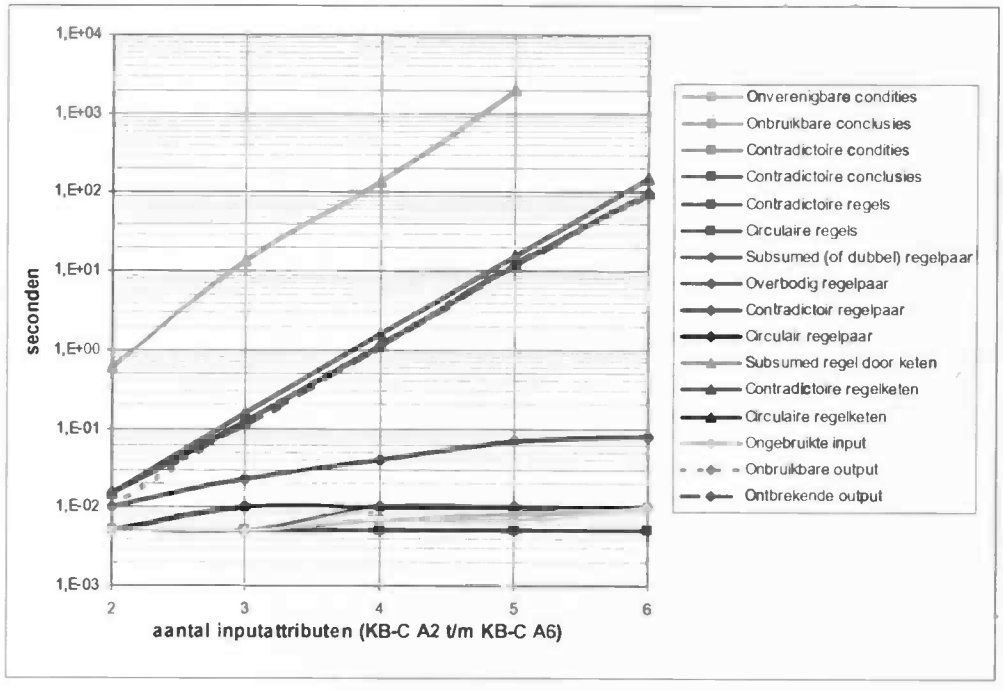
# APPENDIX E
## Originele resultaten en diagrammen

| | KB-B | | | | | | | | | | KB-C | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | rbx20 | rbx40 | rbx60 | rbx80 | rbx100 | rbx120 | rbx140 | rbx160 | rbx180 | rbx200 | A2 | A3 | A4 | A5 | A6 |
| regels | 480 | 960 | 1440 | 1920 | 2400 | 2880 | 3360 | 3840 | 4320 | 4800 | 31 | 62 | 62 | 93 | 93 |
| gem. condities per regel | 2,38 | 2,38 | 2,38 | 2,38 | 2,38 | 2,38 | 2,38 | 2,38 | 2,38 | 2,38 | 1,48 | 1,48 | 1,48 | 1,48 | 1,48 |
| gem. conclusies per regel | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,23 | 1,23 | 1,23 | 1,23 | 1,23 |
| attributen | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 14 | 14 | 21 | 21 |
| inputattributen | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 2 | 3 | 4 | 5 | 6 |
| goalattributen | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 4 | 4 | 6 | 6 |
| environments | 144 | 144 | 144 | 144 | 144 | 144 | 144 | 144 | 144 | 144 | 36 | 216 | 1296 | 7776 | 46656 |
| gem. extensielengte | 2,50 | 2,50 | 2,50 | 2,50 | 2,50 | 2,50 | 2,50 | 2,50 | 2,50 | 2,50 | 4,00 | 4,00 | 8,00 | 8,00 | 12,00 |
| tonen environments | 0,040 | 0,040 | 0,040 | 0,040 | 0,040 | 0,050 | 0,060 | 0,060 | 0,060 | 0,060 | 0,010 | 0,035 | 0,135 | 1,372 | 7,501 |
| tonen regelextensies | 0,931 | 2,190 | 3,235 | 4,316 | 5,318 | 6,399 | 7,521 | 8,713 | 9,574 | 10,775 | 0,020 | 0,140 | 1,242 | 12,759 | 96,699 |
| Onverenigbare condities | 0,130 | 0,581 | 3,250 | 7,551 | 12,978 | 19,910 | 27,900 | 35,542 | 45,700 | 54,408 | 0,005 | 0,005 | 0,007 | 0,007 | 0,010 |
| Onbruikbare conclusies | 0,130 | 1,121 | 3,415 | 6,310 | 9,634 | 14,341 | 19,358 | 25,318 | 32,298 | 39,727 | 0,005 | 0,005 | 0,007 | 0,008 | 0,010 |
| Contradictoire condities | 0,005 | 0,010 | 0,010 | 0,010 | 0,010 | 0,012 | 0,017 | 0,020 | 0,020 | 0,020 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 |
| Contradictoire conclusies | 0,005 | 0,010 | 0,010 | 0,010 | 0,010 | 0,012 | 0,015 | 0,015 | 0,015 | 0,015 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 |
| Contradictoire regels | 0,005 | 0,012 | 0,018 | 0,025 | 0,030 | 0,035 | 0,040 | 0,050 | 0,055 | 0,060 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 |
| Circulaire regels | 0,005 | 0,005 | 0,005 | 0,007 | 0,007 | 0,010 | 0,010 | 0,010 | 0,010 | 0,010 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 |
| Subsumed (of dubbel) regelpaar | 0,820 | 3,320 | 7,890 | 13,710 | 26,668 | 35,974 | | | | | 0,005 | 0,005 | 0,010 | 0,010 | 0,010 |
| Overbodig regelpaar | 0,050 | 0,100 | 0,130 | 0,190 | 0,210 | 0,230 | 0,371 | 0,530 | 0,721 | 0,901 | 0,005 | 0,005 | 0,010 | 0,010 | 0,010 |
| Contradictoir regelpaar | 2,133 | 9,194 | 19,650 | 35,924 | 55,590 | 79,518 | 107,741 | 141,971 | 179,106 | 223,527 | 0,010 | 0,023 | 0,040 | 0,071 | 0,080 |
| Circulair regelpaar | 0,080 | 0,400 | 1,152 | 2,544 | 4,186 | 5,949 | 8,420 | 11,127 | 14,241 | 17,626 | 0,005 | 0,010 | 0,010 | 0,010 | 0,010 |
| Subsumed regel door keten | 359,535 | 3425,500 | | | | | | | | | 0,601 | 13,444 | 138,106 | 1996,731 | |
| Contradictoire regelketen | 0,945 | 2,370 | 3,310 | 4,406 | 5,638 | 6,830 | 7,431 | 8,582 | 9,555 | 10,736 | 0,015 | 0,156 | 1,622 | 15,483 | 152,750 |
| Circulaire regelkelen | 0,951 | 2,223 | 3,285 | 4,487 | 5,638 | 6,840 | 7,472 | 8,593 | 9,544 | 10,766 | 0,015 | 0,120 | 1,141 | 11,878 | 96,299 |
| Ongebruikte input | 0,018 | 0,050 | 0,060 | 0,070 | 0,082 | 0,100 | 0,115 | 0,131 | 0,150 | 0,170 | 0,005 | 0,005 | 0,007 | 0,007 | 0,010 |
| Onbruikbare output | 0,931 | 2,223 | 3,275 | 4,587 | 5,638 | 7,231 | 7,431 | 8,590 | 9,554 | 10,725 | 0,010 | 0,120 | 1,152 | 11,733 | 94,406 |
| Ontbrekende output | 0,931 | 2,220 | 3,355 | 4,455 | 5,468 | 6,855 | 7,550 | 8,653 | 9,514 | 10,866 | 0,015 | 0,114 | 1,142 | 11,948 | 96,879 |

**Tabel E1: Alle originele resultaten**




**Figuur E1: Tests met toenemend aantal regels**




**Figuur E2: Tests met toenemend aantal inputattributen**