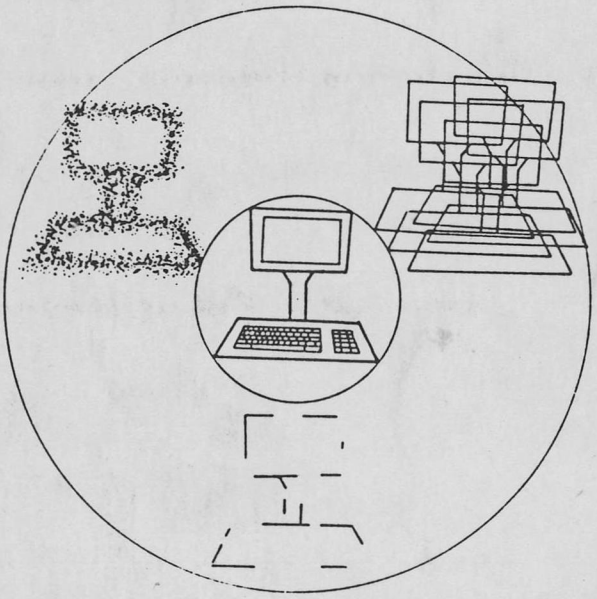
GELOOFWAARDIG ARGUMENTEREN

Formeel onderbouwd

door

N. Roos



Technische Universiteit Delft
Faculteit der Wiskunde en Informatica
Vakgroep Informatica
Deelgroep Theoretische Informatica

GELOOFWAARDIG ARGUMENTEREN

Formeel onderbouwd

door

N. Roos

21 Januari 1987
Begeleiders:
ir R. Sommerhalder
dr C. Witteveen

1. Samenvatting

Met behulp van de predikaat logika kan slechts met volledige en zekere kennis geredeneerd worden. In werkelijkheid is kennis vaak onvolledig, onzeker en daardoor inkonsistent. Om deze redenen zijn verschillende vormen van niet-monotone logika's onderzocht op hun toepasbaarheid voor het redeneren met onvolledige en onzekere kennis. Daar geen van de onderzochte logika's geschikt bleek om de hiervoor opgesomde punten op een formele wijze te hanteren, is een alternatieve logika ontwikkeld. Deze logika gaat uit van de geloofwaardigheid van een bewering. Met dit begrip wordt een ordening van de beweringen aan gegeven, die gebruikt wordt om te bepalen welke bewering verworpen moet worden wanneer we een inkonsistentie in de premissen vinden.

Verder heeft deze logika het voordeel dat er een geschikt boekhoudsysteem bestaat voor het onderhouden van de beweringen, die afgeleid worden. Dit systeem biedt tevens de mogelijkheid om veranderingen in de tijd op eenvoudige wijze te verwerken.

2. Een logika voor onzekere kennis

2.1 Introduktie

Redeneren met onzekere en onvolledige kennis houdt in dat deze kennis vaak inkonsistent is. Immers onzekere kennis, afkomstig van verschillende bronnen -waarnemingen-, kan in tegenspraak zijn. Evenzo kan specifieke kennis in tegenspraak zijn met algemene, maar daardoor minder zekere kennis, de aannamen. Er is in dat geval sprake van een uitzondering.

In een literatuur onderzoek (22) zijn verschillende vormen van niet monotone logika's onderzocht op hun toepasbaarheid voor het redenen met onzekere en onvolledige kennis. De bestudeerde logika's zijn beoordeeld op hun geschiktheid om de volgende aspekten van het redeneren te modelleren.

— Redeneren met behulp van aannamen -verstekredeneren-.

— De geloofwaardigheid van de afgeleide beweringen uitdrukken als een funktie van de geloofwaardigheid van de gebruikte premissen.

— Redeneren met behulp van een mogelijk inkonsistente verzameling premissen zonder dat dit leidt tot: "alles is afleidbaar".

Alleen Reiters default logika (19) bleek voor het redeneren met behulp van aannamen een goede beschrijving te geven. Geen van de logika's bleek geschikt te zijn voor het modelleren van de overige punten.

2.2 Geloofwaardigheid, preferentie en konsistentie

In de logika, die hier beschreven wordt, hebben beweringen een geloofwaardigheid. Deze geloofwaardigheid heeft niets te maken met de kans dat een bewering waar is hoewel er wel een verband is. In deze logika drukt een geloofwaardigheid uit:

> Als twee beweringen p en q in tegenspraak zijn, dan verwerpen we de bewering met de kleinste geloofwaardigheid.

Deze regel geeft dus aan dat we een bewering met een grotere geloofwaardigheid prefereren boven een bewering met een kleinere geloofwaardigheid wanneer de beweringen in tegenspraak zijn. Dit kunnen we op een overeenkomstige wijze ook toepassen op een verzameling inkonsistente beweringen A. We prefereren die konsistente deelverzameling B van A, die de meeste beweringen bevat met de grootste geloofwaardigheid. Zo'n deelverzameling kunnen we selekteren met behulp van van de hierna volgende verwerpingsregel. Deze verwerpingsregel is een algemenere vorm van de hiervoor beschreven verwerpingsregel.

> Een minimale inkonsistente verzameling is een inkonsistente verzameling die door de verwijdering van een bewering konsistent gemaakt kan worden.
>
> Wanneer een verzameling beweringen een minimale inkonsistent verzameling is zodat verwijdering van een der beweringen leidt tot een konsistente verzameling, dan verwijderen we de bewering met de kleinste geloofwaardigheid.

Wanneer we deze verwerpingsregel willen toepassen op een inkonsistente verzameling A dan moeten we een minimale inkonsistente deelverzameling uit A selekteren en daarop de verwerpingsregel toepassen.

Een bewering p kan verworpen worden om een bepaalde inkonsistentie op te heffen. Wanneer vervolgens de verwerping van bewering p overbodig wordt door het opheffen van een andere inkonsistentie, dan moet de bewering p terug geplaatst worden. Dit is noodzakelijk omdat we die konsistente deelverzameling prefereren, die de meeste beweringen met de grootste geloofwaardigheid bevat.

Het is nu noodzakelijk om van elke premisse aan te geven wat de geloofwaardigheid van die premisse is. De premissen waarmee we redeneren, delen we in in klassen. Elke klasse bevat beweringen met een bepaalde geloofwaardigheid. We zullen de geloofwaardigheid aanduiden met een getal uit het gesloten interval [0,1], waarbij 1 de grootste geloofwaardigheid [de ware beweringen] en 0 de kleinste geloofwaardigheid weergeeft. Het voorgaande betekent niet dat er oneindig veel klassen met beweringen zijn. Een getal x uit het interval dient slechts als **naamgeving** voor de klasse C_x. Wel wordt de ordening van de gebruikte klassen door de naam van de klasse bepaald. Als het aantal niet lege klassen, met uitgezondering de klasse C_1, gelijk is aan $n-1$ dan wordt met behulp van de funktie ϕ: $\{1..n\} \rightarrow [0,1]$ een eenduidige afbeelding van een natuurlijk getal i op een naam x van een klasse gerealiseerd. Voor deze afbeelding gelden de relaties $\phi(1)=1$, $\forall x[\phi(i)>\phi(i+1)]$, $\forall i[i \in \{2..n\} \rightarrow C_{\phi(i)} \neq \emptyset]$ en $\forall i[\phi(i)>x>\phi(i+1) \rightarrow C_x=\emptyset]$. In de hierna volgende hoofdstukken zal worden aangetoond dat de logika geen nieuwe klassen introduceert. Het aantal klassen wordt bepaald door het aantal klasse dat gebruikt wordt door de premissen. De klasse C_1 zullen we opvatten als de klasse van ware beweringen. Deze klasse moet daarom konsistent zijn. Dit geldt niet voor de overige klassen.

In de volgende hoofdstukken zullen we soms de notatie p/x in plaats van de notatie $p \in C_x$ gebruiken.

2.3 Inferentieregels en geloofwaardigheid

In het vorige hoofdstuk hebben we gezien wat de betekenis is van de geloofwaardigheid van een bewering. In dit hoofstuk wordt de samenhang tussen de inferentieregels en de geloofwaardigheid van de beweringen, waarop deze regels betrekking hebben, besproken.

Het eerste dat we ons moeten realiseren is dat de geloofwaardigheid van een afgeleide bewering afhankelijk is van de in de afleiding gebruikte beweringen. We zullen dit illustreren aan de hand van een voorbeeld. Dit voorbeeld handelt over een auto die niet meer rijdt.

— De motor loopt niet.

— Als de motor niet loopt dan is het **mogelijk** dat de benzine op is.

konklusie 1 : Het is **mogelijk** dat de bezine op is.

— De bezinemeter staat op nul.

— Als de bezine meter op nul staat dan is het **zeer waarschijnlijk** dat de benzine op is.

konklusie 2 : Het is **zeer waarschijnlijk** dat de benzine op is.

De konklusie in dit voorbeeld heeft twee verschillende geloofwaardigheden. Bij toepassing van de verwerpingsregel is uiteraard alleen de grootste geloofwaardigheid van belang.

Voor de inferentieregels betekent bovenstaande dat de geloofwaardigheid van de consequent van een inferentie regel bepaald wordt door de geloofwaardigheid van de antecedenten. We zullen nu achtereenvolgens de volgende inferentieregels bekijken.

$$\frac{p,q}{p \wedge q} \qquad \frac{p}{p \vee q} \qquad \frac{p,\ p \rightarrow q}{q} \qquad \frac{p \wedge q}{p} \qquad \frac{\neg\neg p}{p}$$

Intuitief koppelen we hieraan de volgende geloofwaardigheidsrelaties

- $$\frac{p/x,\ q/y}{p \wedge q/min(x,y)}$$

  Bijvoorbeeld:

  Het gaat zeker *vriezen*, *Het gaat misschien* sneeuwen

  Het gaat misschien vriezen en sneeuwen

- $\dfrac{p/x}{p \vee q/x}$

Deze regel is misschien niet direkt intuitief duidelijk. Men had dan waarschijnlijk de volgende relatie verwacht. $\dfrac{p/x\,,q/y}{p \vee q/max(x,y)}$ Hierbij wordt echter van meer informatie gebruik gemaakt dan voor de inferentieregel noodzakelijk is. Wanneer over de beweringen p en q beschikken dan krijgen we door twee keer de inferentieregel toe te passen hetzelfde resutaat. De verwerpingsregel maakt immers gebruik van de grootste geloofwaardigheid van een bewering.

- $\dfrac{p/x\,,p \rightarrow q/y}{q/min(x,y)}$

Bijvoorbeeld:

Jan is mogelijk *ziek*, *Als jan ziek is blijft hij zeker* thuis
Jan blijft mogelijk thuis

Het stormt, Als het stormt is een dijkdoorbraak mogelijk
Een dijkdoorbraak is mogelijk

- $\dfrac{p \wedge q/x}{p/x}$

Bijvoorbeeld:

Het is mogelijk *dat het buiten vriest en glad is*
Het is mogelijk dat het buiten glad is

- $\dfrac{\neg\neg p/x}{p/x}$

Bijvoorbeeld:

Het is waarschijnlijk *niet zo dat het niet regent*
Het regent waarschijnlijk

Met behulp van de hiervoor gegeven geloofwaardigheidsrelaties kunnen we de hierna volgende stelling bewijzen. In deze stelling maken we gebruik van de volgende funktie.

De funktie min(N) is gedefinieerd als min(N) = [als N={} dan 1 anders [als $N=\{q_1 \in C_{x_1},\ldots,q_k \in C_{x_k}\}$ dan $\min(\{x_1,\ldots,x_k\})$]]

Stelling:
Als N de verzameling beweringen is, die gebruikt zijn in de afleiding van p en $N=\{q_1 \in C_{x_1},\ldots q_k \in C_{x_k}\}$ dan $p \in C_z$ met $z=\min(\{x_1,\ldots,x_k\})$

Bewijs:
Het bewijs verloopt met behulp van induktie naar het aantal inferentie stappen dat nodig is voor de afleiding van een bewering.

Initialisatiestap: Ieder axioma is afhankelijk van 0 premissen en dus afhankelijk van de omgeving {} Iedere premisse p is afhankelijk van 1 premisse en dus afhankelijk van omgeving $\{p\}$

Induktieveronderstelling: Als bewering p afleidbaar is in j stappen met j<i en in de afleiding van p zijn de premissen uit de verzameling N gebruikt dan $p \in C_{min(N)}$

Induktiestap Voor afleidingsstap i kunnen we de volgende situaties onderscheiden:

- Veronderstel dat p afhankelijk is van de omgeving N en $p \rightarrow q$ afhankelijk is van de omgeving M. Er geldt $p \in C_x$ met x=min(N) en $p \rightarrow q \in C_y$ met y=min(M). Hieruit volgt dat $q \in C_z$ met $z = min(x, y) = min(min(N), min(M)) = min(N \cup M)$
- Veronderstel dat p afhankelijk is van premissen N en q van de premissen M. Er geldt $p \in C_x$ met $x = min(N)$ en $q \in C_y$ met $y = min(M)$. Hieruit volgt dat $p \wedge q \in C_z$ met $z = min(x, y) = min(min(N), min(M)) = min(N \cup M)$.
- Veronderstel dat p afhankelijk is van de premissen N. Er geldt $p \in C_x$ met $x = min(N)$. Hieruit volgt dat $p \vee q \in C_z$ met $z = x = min(N)$
- Veronderstel dat $p \wedge q$ afhankelijk is van de premissen N. Er geldt $p \wedge q \in C_x$ met $x = min(N)$. Hieruit volgt $p \in C_z$ met $z = x = min(N)$.
- Veronderstel dat $\neg\neg p$ afhankelijk is van de premissen N. Er geldt $\neg\neg p \in C_x$ met $x = min(N)$. Hieruit volgt $p \in C_z$ met $z = x = min(N)$.

□

Wanneer gebruik gemaakt wordt van meer premissen dan minimaal noodzakelijk is voor het afleiden van een bewering, heeft dit tot gevolg dat we een geloofwaardigheid voor de bewering kunnen vinden, die lager is dan mogelijk. Er geldt namelijk $N \subset M :\rightarrow: min(N) \geqslant min(M)$

Met behulp van de vorige stelling kunnen we de hierna volgende belangrijke stelling bewijzen. Deze stelling toont aan dat de geloofwaardigheidslogika goed is.

Stelling

Laat een verzameling A en B minimale inkonsistente verzamelingen beweringen zijn, waarvoor geldt $B \subseteq Th(A)$. Er geldt nu:

Als p de kleinste geloofwaardigheid heeft in A dan is er een q, q heeft de kleinste geloofwaardigheid in B en $q \notin Th(A - \{p\})$.

Het maakt dus niet uit of we uit een afgeleide verzameling de minst geloofwaardige bewering verwerpen of uit de oorspronkelijke verzameling beweringen.

Bewijs:

Lemma

$\forall p \in A$ geldt dat er een $q \in B$ is en voor iedere verzameling $R \subset A$: als $R \vdash q$ dan $p \in R$.

Bewijs:

Veronderstel dat er een $p \in A$ is, waarvoor geldt: $\forall q \in B\ \exists R \subset A : [R \vdash q$ *en* $p \notin R]$. Maar dan geldt $B \subseteq Th(A - \{p\})$. Omdat $A - \{p\}$ konsistent is, is dus ook B konsistent.

Tegenspraak

□

Zij de bewering p de minst geloofwaardige bewering uit A. Uit het lemma volgt dat er een bewering $q \in B$ is en dat de bewering $p \in A$ voorkomt in iedere afleiding van q. Er geldt dus $q \notin Th(A - \{p\})$. Omdat p de kleinste geloofwaardigheid in A heeft, moet volgens de vorige stelling de bewering q de kleinste geloofwaardigheid in B hebben. De bewering q kan dus uit B verworpen worden.

□

De volgende stelling kunnen we echter niet bewijzen:

> Als q de kleinste geloofwaardigheid heeft in B dan is er een p, p heeft de kleinste geloofwaardigheid in A en $q \notin Th(A - \{p\})$.

Deze stelling kunnen we niet bewijzen omdat er meerdere afleidingen kunnen bestaan voor een bewering uit B. Er is echter wel altijd een bewering q met de kleinste geloofwaardigheid in B, waarvoor geldt: $\exists p \in A : q \notin Th(A/p)$ en p heeft de kleinste geloofwaardigheid in A. Wanneer echter in de verzameling A slechts een bewering voorkomt, welke de kleinste geloofwaardigheid heeft, dan is het wel mogelijk om de stelling te bewijzen. Dit is mogelijk omdat dan de bewering met de kleinste geloofwaardigheid in B alleen kan worden afgeleid met behulp van de bewering met de kleinste geloofwaardigheid uit A.

Met behulp van de hiervoor bewezen stelling en de verwerpingsregel kan de geloofwaardigheidspropagatie van de volgende inferentieregels bewezen worden.

$\{p/1\} \cup R \vdash q/i$ *en* R *is konsistent* $:\rightarrow: R \vdash (p \rightarrow q)/i$.

$\{p/1\} \cup R \vdash q/i$, $\{p/1\} \cup R \vdash \neg q/j$ *en* R *is konsistent* $:\rightarrow: R \vdash \neg p/min(i,j)$

$\{p/1\} \cup R \vdash r/i$ *en* $\{q/1\} \cup R \vdash r/j$ *en* R *is konsistent* $:\rightarrow: \{(p \vee q)/k\} \cup R \vdash r/min(i,j,k)$

$$\frac{\forall x\ p(x)/i}{p(a)/i}$$

$$\frac{p(a)/i}{\exists x\ p(x)/i}$$

$$\frac{p(a)/i}{\forall x\ p(x)/i}$$ mits a niet voorkomt in de premissen en in $\forall x\ p(x)$.

$\{p(c)/1\} \cup R \vdash q/j$ *en* R *is konsistent* $:\rightarrow: \{\exists x\ p(x)/i\} \cup R \vdash q/min(i,j)$ mits c een nieuwe naam is die niet in q voorkomt.

2.4 Generatieve premissen

We voeren in deze logika nog een nieuw begrip in, namelijk "generatieve premisse". Een generatieve premisse is een bewering, waarin een of meer vrije variabelen voorkomen. Elke bewering, die we verkrijgen door substitutie van termen voor de vrije variabelen, vatten we op als een nieuwe premisse. Deze nieuwe premissen kunnen ieder afzonderlijk verworpen worden, wanneer een tegenspraak wordt afgeleid, zonder dat dit invoed heeft op de generatieve premisse. Met behulp van een generatieve premisse kan een mogelijk oneidige verzameling premissen geintroduceerd worden. Het volgende voorbeeld illusteert het gebruik van een generatieve premisse. Hierbij is van de volgende notatie gebruik gemaakt: $p \in C_x$ wordt genoteerd als p/x.

1. $mus(a)/0.9$
2. $struisvogel(b)/0.9$
3. $\forall x\ [mus(x) \rightarrow vogel(x)]/1.0$
4. $\forall x\ [struisvogel(x) \rightarrow vogel(x)]/1.0$
5. $\forall x\ [vogel(x) \rightarrow kan\_vliegen(x)]/0.8$
6. $\forall x\ [struisvogel(x) \rightarrow \neg kan\_vliegen(x)]/1.0$

Deze verzameling uitspraken is inkonsistent want $kan\_vliegen(b)$ en $\neg kan\_vliegen(b)$ zijn afleidbaar. Om deze verzameling konsistent te maken wordt premisse vijf verworpen. Daardoor

kan uit deze verzameling premissen de bewering *kan_vliegen*(a) niet meer afgeleid worden. Indien we de vijfde bewering vervangen door de generatieve bewering [*vogel*$(x) \rightarrow$ *kan_vliegen*(x)] /0.8, kunnen de beweringen *kan_vliegen*(a) en $\neg$*kan_vliegen*(b) beide afgeleid worden.

2.5 Dekpunt karakterisering

In dit hoofdstuk zullen we de verzameling beweringen, die afleidbaar zijn uit de premissen, definieren met behulp van een dekpunt operator en de monotone afleidingsregels van de predikaat logika. Deze verzameling van beweringen zullen we aanduiden het begrip theorie.

We selekteren een konsistente deelverzameling B van A, waarbij $A = \bigcup_{i=1}^{n} C_{\phi(i)}$ en $B = \bigcup_{i=1}^{n} D_{\phi(i)}$

Hierbij is $D_1 = C_1$ en $D_{\phi(i)}$ voor $i > 1$ een dekpunt van de funktie G_i. Deze funktie is gedefinieerd als:

$$G_i = \lambda\, S\, [\{q \mid \exists p \in C_{\phi(i)}[q \textit{ is een instantie van } p \textit{ en } \{q\} \cup S \cup \bigcup_{k=1}^{i-1} D_{\phi(k)} \textit{ is konsistent}]\}]$$

$$D_{\phi(i)} = G_i(D_{\phi(i)})$$

Met behulp van deze definitie is het mogelijk om meerdere konsistente deelverzamelingen, die we in het vervolg zullen aan duiden met omgeving, uit de premisse te selekteren.

Stelling
Als S een dekpunt is voor G_i en $P \subset S \subset Q$ dan is noch P noch Q een dekpunt van G_i.

Stelling
Als er meer dan 1 dekpunt bestaat voor G_i dan is elke vereniging van 2 dekpunten van G_i inkonsistent.

De nu verkregen verzameling premissen B vormt de wereld, die we voor waar houden. Met behulp van deze wereld kan de theorie T van premissen A gedefinieerd worden.

$$T = Th(B) \quad \text{waarbij } Th(S) = \{p \mid S \vdash p\}$$

Het verband tussen de dekpunten en de verwerpings regel uit hoofdstuk 2.2 wordt weer gegeven door de volgende stellingen.

Stelling
$\forall B$: B is de verzameling die overblijft na het opheffen van alle inkonsistenties (#) uit A met behulp van de verwerpingsregel en $B = \bigcup_{k=1}^{n} D_{\phi(k)} :\rightarrow: D_{\phi(k)}$ is een dekpunt van de funktie G_k.

Bewijs
Het bewijs verloopt met induktie naar het aantal klassen.

Initialisatiestap: $C_\phi(1)$ is konsistent en $C_{\phi(1)} = D_{\phi(1)}$.

Een bewering, die voor het opheffen van een inkonsistentie verworpen werd maar wier verwerping overbodig wordt na het opheffen van een andere inkonsistentie [zie hoofdstuk 2.2] moet weer terug geplaatst worden.

Induktieveronderstelling: $\forall j < i$ [$D_{\phi(j)}$ *is een dekpunt voor funktie* G_j]

Induktiestap: Veronderstel dat $D_{\phi(i)}$ geen dekpunt is voor G_i.
D.w.z. $S_i = G_i(D_{\phi(i)})$ en $D_{\phi(i)} \neq S_i$. Omdat $D_{\phi(i)}$ konsistent is geldt: $D_{\phi(i)} \subset S_i$. Maar dan is er een bewering $q \in S_i$ en $q \notin D_{\phi(i)}$, die verworpen is uit A. Dus $\{q\} \cup D_{\phi(i)} \cup \bigcup_{k=1}^{i-1} D_{\phi(k)}$ is inkonsistent. Omdat $D_{\phi(1)},\ldots,D_{\phi(i-1)}$ dekpunten zijn voor respektievelijke $G_1,\ldots,G_{i-1}$ geldt: $q \notin S_i$.
Tegenspraak

$D_{\phi(j)}$ is een dekpunt voor funktie G_j.

□

Stelling

$\forall B: B = \bigcup_{k=1}^{n} D_{\phi(k)}$ *en* $D_{\phi(k)}$ is een dekpunt van funktie $G_k :\rightarrow:$ B ontstaat uit A door toepassing van de verwerpingsregel.

Bewijs

Het bewijs verloopt met behulp van induktie naar het aantal te verwerpen beweringen. In het bewijs zullen we met B_m de verzameling aanduiden waaruit nog m beweringen verworpen moeten worden om de verzameling B te krijgen.

Initialisatiestap $m = 0$: Er hoeft geen bewering verworpen te worden. $\bigcup_{k=1}^{n} D_{\phi(k)} = B = B_0$

Induktieveronderstelling: $\forall m < j$ [$|B_m - B| = m :\rightarrow:$ de beweringen $B_m - B$ kunnen uit B_m verworpen worden, zodat de beweringen B over blijven.]

Induktiestap: Zij $|B_{m+1} - B| = m+1$, $p \in [B_{m+1} - B]$ en $p \in C_{\phi(i)}$. Uit de definitie van de dekpunten volgt: $\{p\} \cup \bigcup_{k=1}^{i} D_{\phi(k)}$ is inkonsistent. Uit de definitie van een dekpunt volgt dat p de kleinste geloofwaardigheid heeft in een minimale inkonsistente deelverzameling van $\bigcup_{k=1}^{i} D_{\phi(k)}$. Omdat p de kleinste geloofwaardigheid heeft kan bewering p verworpen worden. Door de verwijdering van p uit B_{m+1} krijgen we B_m met $|B_m - B| = m$. Volgens de induktie veronderstelling kunnen de beweringen $B_m - B$ uit B_m verwijderd worden, zodat de beweringen B overblijven.

Zij $A = B_m$ dan kunnen de beweringen, die niet in B voorkomen verworpen worden.

□

2.6 Een model voor de logika

We definieren een interpretatie voor de beweringen op dezelfde wijze als de Tarski semantiek wordt gedefinieerd. Een interpretatie kunnen we beschouwen als een beschrijving van een mogelijke wereld. We zijn geinteresseerd in die wereld, die het beste aansluit bij de premissen. D.w.z. we prefereren een wereld, die beter aansluit bij de premissen uit een klasse met een hogere geloofwaardigheid boven een wereld, die beter aansluit bij de premissen uit de klasse met een lagere geloofwaardigheid. De wereld, die volgens dit kriterium het beste aansluit bij de premissen, duiden we aan als een model voor deze premissen. Een model voor de premissen kunnen we nu als volgt definieren.

Een wereld -interpretatie- M_1 is een model voor de klasse C_1 als geldt: $\forall p \in C_1$: $M_1(p) = true$.

Een wereld M_i is een model voor de klassen $C_{\phi(1)},\ldots,C_{\phi(i)}$ als M_i een model is voor de klassen $C_{\phi(1)},\ldots,C_{\phi(i-1)}$ en voor iedere $p \in C_{\phi(i)}$ met $M_i(p) = false$ geldt dat er geen Tarski model is voor $\{p\} \cup Q(M_i)$.

Hierbij is de afbeelding Q gedefinieerd als :

$$Q(I) = \{p \mid p \in A \text{ en } I(p) = true\}$$

Een model M voor de premissen A met $A = \bigcup_{i=1}^{n} C_{\phi(i)}$ is een model voor de klassen $C_{\phi(1)},...,C_{\phi(n)}$. Dus $M = M_n$

De volgende twee stellingen geven de relatie tussen het hiervoor beschreven model en de dekpunten uit hoofdstuk 2.5.

Stelling

$\forall B\ \exists M : [\forall p \in B : M(p) = true, B = \bigcup_{k=1}^{n} D_{\phi(k)}$ *en* $D_{\phi(k)}$ *is een dekpunt van* $G_k :\rightarrow: M$ *is een model voor de premissen* A]

Bewijs

Het bewijs verloopt met behulp van induktie naar het aantal klassen.

Initialisatiestap: M_1 is een Tarski model voor $D_{\phi(1)}$ en $D_{\phi(1)} = C_{\phi(1)}$. Hieruit volgt dat M_1 een model is voor $C_{\phi(1)}$.

Induktieveronderstelling: $\exists M_{i-1} : M_{i-1}$ *is een Tarski model voor* $\bigcup_{k=1}^{i-1} D_{\phi(k)} :\rightarrow:$

M_{i-1} *is een model voor* $\bigcup_{k=1}^{i-1} C_{\phi(k)}$.

Induktiestap: $\bigcup_{k=1}^{i} D_{\phi(k)}$ is konsistent. Er bestaat dus een Tarski model M_i voor $\bigcup_{k=1}^{i} D_{\phi(k)}$.

Veronderstel dat M_i geen model is voor $\bigcup_{k=1}^{i} C_{\phi(k)}$.

Volgens de induktie veronderstelling is M_i wel een model voor $\bigcup_{k=1}^{i-1} C_{\phi(k)}$. Dus is er een $p : p \in C_{\phi(i)}, p \notin D_{\phi(i)}$ en er is een Tarski model I voor $\{p\} \cup \bigcup_{k=1}^{i} D_{\phi(k)}$. Uit de definitie van een dekpunt volgt, dat $\{p\} \cup \bigcup_{k=1}^{i} D_{\phi(k)}$ inkonsistent is. Er is dus geen Tarski model voor $\{p\} \cup \bigcup_{k=1}^{i} D_{\phi(k)}$.

Tegenspraak.

M_i in een model voor $C_{\phi(1)}..C_{\phi(i)}$.

□

Stelling.

$\forall M\ \exists B : [M$ *is een model voor de premissen* $A :\rightarrow: \forall p \in B : M(p) = true, B = \bigcup_{k=1}^{n} D_{\phi(k)}$ *en* $D_{\phi(k)}$ *is een dekpunt van* G_k]

Bewijs

Het bewijs verloopt met induktie naar het aantal klassen.

Intialisatiestap: M_1 is een model voor $C_{\phi(1)}$. $D_{\phi(1)} = C_{\phi(1)}$ en $C_{\phi(1)}$ is konsistent. Dus M_1 is een Tarski model voor $D_{\phi(1)}$.

Induktieveronderstelling: M_{i-1} is een model voor $\bigcup_{k=1}^{i-1} C_{\phi(k)} :\rightarrow:$

$\exists B : [B = \bigcup_{k=1}^{n} D_{\phi(k)}$ *en* M_{i-1} *is een Tarski model voor* $\bigcup_{k=1}^{i-1} D_{\phi(k)}$

Induktiestap: Veronderstel M_i is een model voor $\bigcup_{k=1}^{i} C_k$. Zij $\bigcup_{k=1}^{i} S_k$ met $S_k \subseteq C_k$ de beweringen waarvoor geldt: $\forall p \in S_k : M_i(p) = true$. Volgens de induktieveronderstelling $\exists B$ *met* $S_k = D_{\phi(k)}$ *voor* $k < i$. Veronderstel dat S_i geen dekpunt van G_i is.

Dit betekent: $\exists p : p \in S_i$ *en* $p \notin G_i(S_i)$ *of* $\exists p : p \in G_i(S_i)$ *en* $p \notin S_i$.

De eerste situatie is onmogelijk omdat uit $p \in S_i$ en de definitie van een dekpunt volgt dat $\{p\} \cup S_i \cup \bigcup_{k=1}^{i-1} D_{\phi(k)}$ konsistent is. Dus $p \in G_i(S_i)$.

De tweede situatie is eveneens onmogelijk omdat uit de definitie van G_i volgt dat $\{p\} \cup S_i \cup \bigcup_{k=1}^{i-1} D_{\phi(k)}$ konsistent is. Maar dan is er een Tarski model I voor $\{p\} \cup S_i \cup \bigcup_{k=1}^{i-1} D_{\phi(k)}$.

Tegenspraak.

□

2.7 Bijzondere eigenschappen

Deze logika heeft een aantal bijzondere eigenschappen.

1. In deze logika is de deduktie stelling niet zonder meer te gebruiken. Dit komt omdat $p \wedge \neg p \rightarrow q$ niet equivalent is aan $\{p, \neg p\} \mid\sim q$. In deze logika wordt immers slechts met een konsistente deelverzameling van de premissen geredeneerd.
2. In deze logika is de implikatie niet omkeerbaar in een generatieve bewering. D.w.z. $p(x) \rightarrow q(x)$ is niet equivalent aan $\neg q(x) \rightarrow \neg p(x)$

   Bijvoorbeeld:
   "bloemen bloeien in de lente" is niet equivalent aan "wat niet in de lente bloeit is geen bloem"
3. In deze logika zijn intuitief niet korrekte resultaten afleidbaar. We zullen dit illustreren met het volgende voorbeeld.
   1. $\forall x$ [*mus*$(x) \rightarrow$ *vogel*(x)]/1.0
   2. $\forall x$ [*struisvogel*$(x) \rightarrow$ *vogel*(x)]/1.0
   3. [*vogel*$(x) \rightarrow$ *kan_vliegen*(x)]/0.8
   4. $\forall x$ [*struisvogel*$(x) \rightarrow \neg$*kan_vliegen*(x)]/1.0

   Zonder een bewering *struisvogel*(*jan*) aan deze premissen toe te voegen kunnen we de beweringen $\forall x$ [*struisvogel*$(x) \rightarrow$ *kan_vliegen*(x)] en $\forall x$ [*struisvogel*$(x) \rightarrow \neg$*kan_vliegen*(x)] afleiden. Deze intuitief niet korrekte resultaten kunnen we voorkomen door een alternatieve implikatie te gebruiken. Deze alternatieve implikatie zouden we als volgt kunnen definieren:

   $p(x) \rightarrow\!\!\!\!\rightarrow q(x)$, waarbij $p(x)$ een welgevormde formule is, waarin de variabelen x vrij voorkomen $:\longleftrightarrow :\exists x\ p(x)$ is een premisse en de formule $p(x) \rightarrow\!\!\!\!\rightarrow q(x)$ vervangen we door de formule $p(x) \rightarrow q(x)$

Bovenstaande definitie geeft aan dat deze implikatie alleen zinvolle relaties beschijft. De bewering $\forall x\,[hond\,(x\,) \wedge kan\_kraaien\,(x\,) \rightarrow\!\rightarrow kan\_vliegen\,(x\,)]$ is geen zinvolle relatie omdat er geen honden bestaan, die kunnen kraaien.

Het gebruik van de implikatie vraagt om een nader onderzoek.

3. Relaties met andere systemen

3.1 Verstek redeneren

Zoals in de introduktie van hoofdstuk 2 al is opgemerkt, bleek uit het literatuur onderzoek dat alleen Reiter's versteklogika (19) geschikt is om het redeneren met behulp van aannamen te beschrijven. De versteklogika, die alleen gebruik maakt van de normale verstekregels, kan op eenvoudige wijze met de geloofwaardigheidslogika beschreven worden. Hiervoor hebben we slechts twee klassen nodig, namelijk C_1 en C_x met $x < 1$. Door de verzameling gesloten premissen W in de klasse C_1 te plaatsen en de verzameling verstekregels D te vervangen door een verzameling generatieve premissen en deze in klasse C_x te plaatsen, krijgen we een equivalente theorie in de geloofwaardigheidslogika. Een verstekregel van de vorm $\frac{p(x) : Mq(x)}{q(x)}$ wordt hier vervangen door de generatieve premisse $p(x) \rightarrow q(x)$. De extensies van de versteklogika komen nu overeen met de werelden, waarin men kan geloven, in de geloofwaardigheidslogika.

Volgens Reiter (19) zijn alle verstekregels, die wij doorgaans gebruiken, normale verstekregels. In een later artikel, dat Reiter samen met Criscuolo schreeft (20), komt hij hierop terug. Interakties tussen normale defaults onderling en met de implikatie, kan tot ongewenste gevolgen leiden. Deze gevolgen zijn onbedoelde inkonsistenties en transitieve relaties. Van dit laatste geven we een voorbeeld:

1. De meeste studenten zijn volwassen. $\frac{p(x) : Mq(x)}{q(x)}$

2. De meeste volwassenen heben een baan. $\frac{q(x) : Mr(x)}{r(x)}$

Hieruit volgt: De meeste studenten hebben een baan.

Om deze ongewenste interaktie te voorkomen moet gebruik gemaakt worden van niet normale verstekregels.

$$\frac{q(x) : M(\neg p(x) \wedge r(x))}{r(x)}$$

In het artikel laten zij zien dat deze verstekregels herschreven kunnen worden naar normale verstekregels. Deze herschrijving is afhankelijk van het soort interaktie tussen de oorspronkelijke verstekregels. Voor ons voorbeeld wordt dit:

$$\frac{p(x) : Mq(x)}{q(x)}, \frac{q(x) : M \neg p(x)}{\neg p(x)} \text{ en } \frac{p(x), q(x) : Mr(x)}{r(x)}.$$

Bij het gebruik van niet normale verstekregels evenals bij de herschrijving naar normale verstekregels, introduceren we een specifiekere beschrijving dan oorspronkelijk bedoeld was. In feite zijn we uitzonderingen op de oorspronkelijke verstekregel aan het beschrijven. Wanneer we de premissen uit het voorbeeld uitbreiden met de regel:

3. bejaarden zijn volwassen.

krijgen we nog een uitzondering. Hierdoor moeten de regels opnieuw herschreven worden. Het beschrijven van uitzonderingen op verstekregels stemt niet overeen met de bedoeling van een verstekregel.

In de geloofwaardigheidslogika kunnen we, bij elke ongewenste interakie, de oorspronkelijke verstekregels handhaven, eventueel aangevuld met een extra verstekregel. Dit is in deze logika mogelijk, door de juiste geloofwaardigheid aan de beweringen toe te kennen. Voor het voorbeeld wordt dit:

1. $[p(x) \rightarrow q(x)]/x$
2. $[q(x) \rightarrow r(x)]/y$
3. $[p(x) \rightarrow \neg r(x)]/z$

waarbij $y \leqslant x$ en $y \leqslant z$.

We hebben de verstekregel:

De meeste student hebben geen baan.

toegevoegd. Deze oplossing voldoet aan onze intuitie. Hiermee hebben we dus een systeem dat voldoet aan de eisen die we gesteld hebben voor redeneren met aannamen en onzekere kennis.

3.2 Truth Maintenance System

Voor een implementatie van de geloofwaardigheidslogika is ATMS van J de Kleer (8) zeer geschikt. In ATMS wordt een premissen gerepresenteerd met behulp van een aanname en een veronderstelling. Voor elke premisse p plaatsen we in ATMS een aanname P, een veronderstelling p en de rechtvaardiging $P \Longrightarrow p$. Om de klasse weer te geven waarin een bewering zich bevindt en om generatieve premissen te representeren moet ATMS wel enigzins aangepast worden. De aangepaste versie wordt aangeduid met ATMS*.

Bij elke minimale omgeving van een bewering hoort een bepaalde geloofwaardigheid, die bepaald wordt door het minimale aantal premissen, dat nodig is in de afleiding van de bewering. We moeten dus bij elke aanname een klasse aangeven. Hiermee kan de klasse van de afgeleide beweringen bepaald worden.

De generatieve premissen worden anders dan de overige premissen behandeld, omdat eerstgenoemde premissen niet verworpen worden, wanneer een tegenspraak afgeleid wordt. Instanties van deze premissen kunnen wel verworpen worden. Omdat instanties van een generatieve premisse verworpen kunnen worden, plaatsen we voor elke instantie een aanname in ATMS*. Deze aanname kan binnen ATMS* op gelijke wijze als de overige aannamen verwerkt worden. Met behulp van van een indikator kunnen we aangeven, dat deze aanname een instantie is van een generatieve premisse. De generatieve premisse kunnen we representeren met behulp van een generatieve knoop. Deze knoop bevat de generatieve premisse en een lijst met verwijzingen naar de aanname voor de instanties van de generatieve premisse. Bij de aanname plaatsen we een verwijzing naar de bijbehorende generatieve knoop. Het is de taak van het redeneersysteem om nieuwe aannamen voor de gegenereerde instanties van de generatieve premisse toe te voegen.

ATMS* moet verder in staat zijn alle mogelijke werelden, die voort komen uit een verzameling premissen, te bepalen. Gegeven een verzameling premissen, kunnen de werelden, waarin men kan geloven, bepaald worden met behulp van de verzameling "nogood".

Een belangrijk aspekt, wat tot nu toe buiten beschouwing gelaten is, is de verandering van de premissen in de tijd. Er kunnen nieuwe beweringen aan een klasse worden toegevoegd. Verder kan uit elke klasse, behalve de klasse C_1, een bewering verwijderd worden of naar een andere klasse verhuisd worden. Dit laatste is niet mogelijk met de klasse C_1 omdat deze klasse alleen ware beweringen bevat. Het is daarom verstandig om deze klasse slechts voor de axioma's te gebruiken. De veranderingen in de premissen kunnen ook met ATMS* verwerkt worden. Wanneer een premisse verwijderd wordt moet ATMS* de aanname, behorende bij deze bewering, en de omgevingen in de labels, waarin die aanname voorkomt, verwijderen. Het toevoegen van een premisse geeft ook geen problemen. ATMS* plaatst een nieuwe aanname voor de premisse en een veronderstelling indien de bewering nog niet voorkomt. Vervolgens voegt men een rechtvaardiging toe voor de premisse. De verandering van klasse kan men realiseren door de klasse van de aanname te veranderen.

4. Een kennissysteem gebaseerd op geloofwaardigheislogika

Een kennissysteem, dat redeneert met onzekere kennis kunnen we op de hiervoor behandelde logika baseren. Zo'n kennissysteem bestaat uit twee onderdelen, namelijk een boekhoudsysteem en een redeneersysteem.

4.1 Het boekhoudsysteem

Het boekhoudsysteem registreert de samenhang tussen de beweringen. De context, waarin we geinteresseerd zijn, verandert door het afleiden van tegenspraken of door het toevoegen van nieuwe premissen en het verwijderen van andere premissen. Dit boekhoudsysteem kan gebaseerd worden op de ATMS van J. de Kleer (8). Het boekhoudsysteem houdt bij, welke beweringen in de verschillende contexten gelden. Omdat ATMS bijhoudt welke beweringen in welke contexten gelden, kunnen we gemakkelijk van context wisselen. Alle resultaten, die het redeneersysteem ooit heeft afgeleid en die in de nieuwe context gelden, kunnen we onmiddellijk gebruiken.

4.2 Het redeneersysteem

Dit redeneersysteem bestaat uit twee onderdelen, namelijk een deductiesysteem en een besturingssysteem. Het deductiesysteem is onafhankelijk van de kennis waarover geredeneerd wordt. Dit systeem tracht slechts bewijzen te vinden voor beweringen. Het besturingssysteem bepaalt echter welke beweringen we proberen te bewijzen, met welke premissen we redeneren en wanneer om externe informatie gevraagd wordt. Het besturingssysteem is dan ook toepassings afhankelijk. Dit besturingssysteem kan opgevat worden als een metalogika. Deze metalogika kan eveneens met onzekere kennis beschreven worden. De predikaten, die in de metalogika gebruikt worden, hebben betrekking op het deductiesysteem en de data waarover geredeneerd wordt.

Wanneer het besturingssysteem gerealiseerd wordt als een metalogika, die gebaseerd is op de geloofwaardigheidslogika, dan kan de metalogika ook met een ATMS gerealiseerd worden. Wanneer we in staat zijn om de in de metalogika slechts van een beperkt aantal predikaten gebruik te maken, dan kunnen we het besturingssyteem van de metalogika realiseren met behulp van een toepassings onafhankelijk algorithme. De metalogika moet in dat geval over een van te voeren vastliggend aantal predikaten beschikken, voor onder andere het lezen van interne en externe informatie, het zenden van informatie naar de buitenwereld, het maken van dedukties en het bepalen van de wereld waarin geredeneerd wordt.

Voor het redeneersysteem kunnen we een aantal algemene heuristische regels geven.

— Wanneer het redeneersysteem een bewering bewezen heeft, moet het redeneersysteem nagaan of geen tegenspraak afleidbaar is met de gebruikte premissen.

— Het is niet noodzakelijk om alle mogelijke tegenspraken af te leiden. We kunnen volstaan met na te gaan of een tegenspraak afleidbaar is in de wereld waarin op een bepaald moment redeneerd wordt.

— Het is niet noodzakelijk om alle mogelijke afleidingen voor een bewering te vinden. Er hoeft pas naar een nieuwe, meer geloofwaardige, afleiding gezocht te worden als een van de gebruikte premissen verworpen wordt.

4.3 Een ontwerp voor een implementatie m.b.v ATMS

Dit hoofdstuk behandelt een globaal ontwerp voor de aangepaste ATMS uit hoofdstuk 3.2. Voor een beschrijving van ATMS verwijzen we naar het literatuur onderzoek (22) of naar (8). Van deze ATMS* zullen achtereenvolgens de datastrukturen, de interface naar het redeneer systeem en de primaire funkties behandeld worden.

4.3.1 De datastrukturen

In ATMS* onderscheiden we drie soorten datastrukturen. Deze datastrukturen zullen nu achtereenvolgens behandeld worden.

— In ATMS* onderscheiden we drie soorten knopen namelijk aannamen, generatieve knopen en afgeleide knopen. Deze knopen representeren beweringen, die slechts eenmaal in ATMS* gerepresenteerd mogen worden.

Alle afgeleide knopen hebben een label. Dit label geeft aan welke verzameling aannamen gebruikt is om deze knoop af te leiden. Met deze verzamelingen aannamen moeten de relaties "vereniging" en "deelverzameling van" gerealiseerd worden. Deze relaties kunnen efficient geimplementeerd worden met behulp van bitvektoren. Omdat het aantal aannamen van te voren niet bekend is, maken we gebruik van een lijst van bitvektoren. De grootte van een bitvektor uit deze lijst wordt bepaald door de woordgrootte van de computer. Om te kunnen bepalen, welke aannamen gerepresenteerd worden door een bit uit de bitvektorlijst, moeten we de samenhang registreren. Dit kan door bij elke aanname aan te geven door welk bit deze aanname gerepresenteerd wordt. Om een aanname, die hoort bij een bit, sneller te kunnen opzoeken kunnen we verder nog gebruik maken van een twee-dimensionale tabel om de aanname, die hoort bij een bit, op te zoeken. Deze tabel bestaat uit een lijst met rijen. Elke rij specificeert een eeneenduidig afbeelding van de bits van een bitvektor uit de bitvektorlijst naar de bijbehorende aannamen. Deze afbeelding kan gerealiseerd worden met behulp van verwijzingen.

Een label bezit een variabel aantal verzamelingen met aannamen. Dit betekent dat het label alleen geimplementeerd kan worden met behulp van een lijst van verzamelingen aannnamen.

Omdat de klasse van een premisse slechts af en toe zal veranderen, in ieder geval niet tijdens het redeneerproces, kunnen we de klasse, die hoort bij een minimale omgeving uit het label, bij deze omgeving registreren. Dit maakt het mogelijk om de klasse van een bewering snel te bepalen.

Iedere aanname heeft een bitvektorlijst waarmee het bit, dat behoort bij een aanname, geidentificeerd kan worden. Verder geven we bij een aanname aan of deze gegenereerd is m.b.v een generatieve knoop. Indien de aanname met behulp van een generatieve knoop gegenereerd is, plaatsen we tevens nog een verwijzing in de aanname naar de generatieve knoop, die de generatieve premisse representeert. De generatieve knoop bevat een lijst met verwijzingen naar de aannamen, die de gegenereerde instanties van de generatieve premisse representeren.

De knopen in ATMS* kunnen in een lijst geplaatst worden. Om de verschillende soorten knopen te onderscheiden, maken we gebruik van drie lijsten. Gebruik van verschillende lijsten verhoogt de efficientie wanneer we in een specifiek type knoop geinteresseerd zijn. Wanncer we een ordening voor de beweringen definieren, kan i.p.v. een lijst, ook gebruik gemaakt worden van binaire zoekbomen. Hiermee kunnen we een bepaalde knoop sneller opzoeken.

— Een tweede belangrijke datastruktuur van ATMS* vormen de rechtvaardigingen. Deze rechtvaardigingen, die de samenhang tussen de knopen weergeven, zijn altijd korrekt en hoeven ook nooit veranderd te worden. Om deze reden kunnen de rechtvaardigingen het eenvoudigste met behulp van verwijzingen weergegeven worden. In een knoop, die in de consequent van een rechtvaardiging voorkomt, plaatsen we een lijst met verwijzingen naar de antecedent knopen in een lijst met rechtvaardigingen voor deze knoop. In een antecedent knoop van een rechtvaardiging plaatsen we een verwijzing naar de consequent knoop in een lijst met knopen, die met behulp van deze knoop gerechtvaardigd worden. Elke afgeleide knoop heeft dus een lijst met rechtvaardigingen voor de knoop en een lijst met verwijzingen naar knopen die met behulp van deze knoop gerechtvaardigd worden. Aannamen hebben alleen een lijst met verwijzingen naar knopen, die met behulp van de aannamen gerechtvaardigd worden.

— De derde datastruktuur van ATMS* is de verzameling "nogood". Voor deze verzameling gebruiken we de zelfde representatie als voor een label, omdat deze verzameling dezelfde soort gegevens representeert.

4.3.2 Interface naar het redeneer systeem

Het redeneersysteem moet over de volgende funkties beschikken om met ATMS* te kunnen redeneren.

— Het redeneersysteem moet na kunnen gaan welke beweringen uit een bepaalde klasse in een wereld voor waar gehouden worden.

— Het redeneersysteem moet kunnen opvragen in welke werelden het kan geloven, gegeven een verzameling premissen.

— Het redeneersysteem moet een premisse kunnen toevoegen.

— Het redeneersysteem moet een afgeleide knoop kunnen toevoegen.

— Het redeneersysteem moet een rechtvaardiging kunnen toevoegen.

— Het redeneersysteem moet instanties van een generatieve premisse kunnen toevoegen.

— Het redeneersysteem moet de klasse van een premisse kunnen veranderen.

— De premissen, die niet meer van belang zijn, moet het redeneersysteem kunnen laten verwijderen. Alle knopen, die alleen met deze premisse afgeleid kunnen worden, kunnen dan ook verwijderd worden. Deze laatste funktie kan gekombineerd worden met een garbage collector. Deze garbage collector verwijdert alle knopen, welke een leeg label hebben en de knopen die slechts met behulp van deze knopen afleidbaar zijn. De enige uitzondering hierop is de knoop "false".

4.3.3 Primaire funkties

De primaire funkties worden door de ATMS* gebruikt om de toestand van de data van ATMS* bij te werken. De funkties van de interface kunnen of gerealiseerd worden door het lezen van de data van ATMS* of met behulp van de primaire funkties. De primaire funkties zijn:

1. Het toevoegen van een nieuwe aanname.
2. Het toevoegen van een generatieve knoop.
3. Genereren van een instantie van een generatieve premisse.
4. Het toevoegen van een nieuwe afgeleide knoop.
5. Het veranderen van de klasse van een aanname of generatieve knoop en het bijwerken van de klasse van de beweringen, die met deze aanname zijn afgeleid.
6. Het toevoegen van een nieuwe rechtvaardiging voor een bewering.
7. Het bijwerken van de labels van de knopen.
8. Het verwerken van een afgeleide tegenspraak. D.w.z.: het bijwerken van de verzameling "nogood", het bijwerken van de labels van de bewering.
9. Het bepalen van de mogelijke werelden, waarin men kan geloven, gegeven een verzameling premissen.

De hiervoor opgesomde funkties worden hieronder gedetailleerder beschreven.

Ad 1. Het toevoegen van een aanname aan ATMS*.

— Reserveer een bit in de bitvektorlijst voor deze aanname.

— Plaats een nieuwe knoop in de lijst met aannamen. Deze knoop bevat de volgende elementen.

- De bewering, die de knoop representeert.
- Een bitvektorlijst om het bit, dat behoort bij deze aanname te identificeren.
- Een veld voor de verwijzing naar de knoop, die met deze aanname gerechtvaardigd wordt.
- Een indikatie of deze premisse een instantie van een generatieve knoop is. Wanneer de premisse een instantie van een generatieve premisse is, dan ook een verwijzing naar de korresponderende generatieve knoop.
- De geloofwaardigheid van de premisse.

— De tabel voor het opzoeken van een aanname, die behoort bij een bit uit de bitvektorlijst, aanpassen voor de nieuwe aanname.

— Indien er nog geen afgeleide knoop is voor dezelfde bewering als de aanname, voeg dan een afgeleide knoop toe voor deze bewering.

— Het plaatsen van een verwijzing in de aanname naar de korresponderende afgeleide knoop.

— Kreeer een omgeving, die alleen de aanname bevat.

— Roep voor de korresponderende afgeleide knoop de procedure voor het bijwerken van een label aan met als argument de hiervoor gekreeerde omgeving.

Ad 2. Het toevoegen van een generatieve premisse.

— Plaats een nieuwe knoop in de lijst met generatieve knopen. Deze knoop bevat de volgende elementen.

- De generatieve premisse, die de knoop representeert.
- Een nog lege lijst voor de verwijzingen naar de instanties van de generatieve premisse.
- De geloofwaardigheid van de generatieve premisse.

Ad 4. Het toevoegen van een nieuwe afgeleide knoop.

— Een nieuwe knoop in een lijst met afgeleide knopen plaatsen. Deze knoop bevat de volgende elementen.

- De bewering, die de knoop representeert.
- Een nog leeg label behorende bij de knoop.
- Een nog lege lijst voor de knopen, die met deze knoop gerechtvaardigd worden.
- Een nog lege lijst voor de rechtvaardigingen van de knoop.

Ad 6. Het toevoegen van een rechtvaardiging.

— Bepaal de consequent en de antecedent knopen.

— Plaats in elke antecedent knoop een verwijzing naar de consequent knoop, in de lijst met met knopen, die met behulp van de knoop gerechtvaardigd worden.

— Plaats bij de consequent knoop een lijst met verwijzingen naar de antecedent knopen in de lijst met rechtvaardigingen voor de knoop.

— Kreeer de nieuwe omgevingen voor de consequent knoop.

— Roep voor de consequent knoop de procedure voor het bijwerken van een label aan. De argumenten van deze procedure zijn de nieuwe omgeving voor de consequent knoop en een verwijzing naar de consequent knoop.

Ad 7. Het bijwerken van het label van een knoop.

Deze funktie wordt aan geroepen met als argumenten een verzameling nieuwe omgevingen voor een knoop en een verwijzing naar die knoop.

— Verwijder uit verzameling omgevingen de omgevingen, die een superverzameling zijn van een andere omgeving.

— Verwijder uit de verzameling die omgevingen, die een superverzameling zijn van een omgeving uit de verzameling nogood.

— Voeg de elementen, die geen superverzameling zijn van een reeds aanwezige omgeving, toe.

— Verwijder de omgevingen uit het label, die een superverzameling zijn van een toegevoegde omgeving.

— Indien er nieuwe omgevingen aan het label zijn toegevoegd, bepaal dan voor iedere knoop, die met deze knoop gerechtvaardigd wordt, de omgevingen, die ontstaan door de nieuwe omgeving van deze knoop. Hiervoor hebben we alleen de rechtvaardigingen nodig, waarin deze knoop als antecedent voor komt.

— Wanneer de verzameling nieuwe omgevingen voor een knoop, die met deze knoop gerechtvaardigd wordt, niet leeg is, herhaal dan deze procedure voor die knoop.

— Wanneer er nieuwe omgevingen aan het label van de knoop "false" zijn toegevoegd, roep dan de procedure "werk de verzameling nogood bij" aan.

Ad 8. Het bijwerken van de vezameling nogood.

— Voeg de omgeving uit het label van de knoop "false", die geen superverzameling is van een reeds aanwezige omgeving, toe aan de verzameling nogood.

— Verwijder uit de verzameling nogood die omgevingen, die een superverzameling zijn van de toegevoegde omgevingen.

— Verwijder uit het label van elke knoop, die omgevingen, die voorkomen in het label van de knoop "false".

De hiervoor beschreven funkties bevatten geen kontroles op ongeoorloofd gebruik van een funktie, zoals het toevoegen van een aanname van een premisse, die al aanwezig is. Deze kontrole zouden de verwerking onnodig vertragen, omdat het het redeneersysteem "weet" wat geoorloofd is. Dit betekent wel dat het redeneersysteem zorgvuldig ontworpen moet worden.

4.4 Toepassing van het kennissysteem op robot besturing

De hiervoor beschreven kennissysteem stelt ons in staat te redeneren met behulp van onzekere kennis. We kunnen dit kennissysteem toepassen bij het besturen van een robot. Het kennissysteem bezit dan onzekere kennis over de omgeving van de robot. Aan de hand van deze kennis kan het kennissysteem bepalen, welke aktie de robot kan uitvoeren. Om de gevolgen van de aktie te evalueren en om vervolg akties te kunnen plannen is het noodzakelijk dat met een beschrijving van een *hypothetische situatie* geredeneerd kan worden. Een methode, waarbij het frame probleem niet optreedt, is gebaseerd op het verwijderen van beweringen en het toevoegen van andere beweringen. Deze methode wordt onderandere toegepast door R.H.Davis & M.Comacho (5), Warren (3), en W.Bibel (1). Bibel doet dit in een eenigzins andere vorm. Toepassing van deze methode is in ons systeem niet zondermeer mogelijk. Het is namelijk niet mogelijk om afgeleide beweringen te verwijderen. Deze beweringen kunnen immers weer opnieuw worden afgeleid. Het is dus noodzakelijk om een van de premissen waarmee deze bewering is afgeleid te verwijderen. De selektie van de premisse, die verwijderd moet worden, is een probleem dat nader onderzocht moet worden. We zullen dit verduidelijken aan de hand van het volgende voorbeeld.

> Een redeneersysteem bepaalt de positie van de robot (leidt zijn positie af) met behulp van kennis over de waargenomen omgeving. Vervolgens bepaalt het redeneersysteem een verplaatsing van de robot om een bepaald doel te bereiken. Om de gevolgen van de geplande

aktie te evalueren, moet het redeneersysteem met de hypothetische situatie, die ontstaat na de uitvoering van de aktie, verder redeneren. Hiertoe moet de oorspronkelijke positie van de robot tijdelijk buiten beschouwing gelaten worden. Dit is alleen mogelijk wanneer die positie niet meer afleidbaar is. Het is duidelijk dat in deze situatie de premissen die, betrekking hebben op de waargenomen omgeving, buiten beschouwing gelaten moeten worden. Dit is onafhankelijk van de geloofwaardigheid van deze premissen.

Rescher geeft in zijn boek "Hypothetical Reasoning" (21) een analyse van deze problematiek. De oplossing, die hij voorstelt, is gebaseerd op een indeling van onze kennis in klassen. Deze klassen ordenen de informatie naar hun "algemeenheid". Wanneer een hypothese in tegenspraak is met onze huidige kennis dan, moet volgens Rescher de algemene kennis de voorkeur hebben boven de specifieke kennis. Met behulp van deze regel kan op korrekte wijze met hypothetische situaties geredeneerd worden.

Een alternatieve benadering voor het plannen van akties, waarbij tevens expliciet een plan gegenereerd wordt, is gebaseerd op de dynamische logika. De logika voor onzekere kennis, die hiervoor beschreven is, is gebaseerd op de eerste orde predikaat logika. Het is echter geen probleem om in plaats van de predikaat logika gebruik te maken van de dynamische logika. S. J. Rosenschein geeft in het arkikel "Plan Synthesis: a logical perspective" (23) een beschrijving hoe de dynamische logika gebruikt kan worden voor het plannen van akties. Ook op dit punt is nader onderzoek vereist.

5. Een prototype voor het kennis syteem

In dit hoofdstuk wordt een prototype voor het kennissysteem behandeld. Dit prototype welke geschreven is in prolog bestaat uit twee onderdelen namelijk een prototype voor ATMS* en een programma dat als redeneersysteem funktioneert. Dit redeneer systeem kreeert alle mogelijke afleidingen van een bepaalde lengte. Er wordt dus een brute-force deductie toegepast. Hiervoor is gekozen omdat nog geen onderzoek gedaan is naar bruikbare strategieen. Bij een demonstratie van de logika voor onzekere kennis wilden we er tevens zeker van zijn dat het prototype de in principe ideale situatie weergaf, waarin alle van belang zijnde tegenpraken gevonden worden. De enige beperking van het prototype is, dat het maximum aantal afleidingsstappen te weinig kan zijn.

Voor de eenvoud van het te realiseren redeneersysteem, wordt geredeneerd met aangepaste hornclauses. De hornclauses die we zullen gebruiken zijn van de volgende vorm:

$$p :- q_1,...,q_n .$$

waarbij $p,q_1,...,q_n$,in tegenstelling tot normale hornclauses, ook negatieve literals kunnen bevatten. De ontkenning van een predikaat in een negatief literal zullen we aangeven met het symbool "~". Deze aangepaste hornclauses zijn ingevoerd om op eenvoudige wijze tegenspraken te kunnen afleiden. Voor het afleiden van tegenspraken introduceren we verder nog het volgende axioma.

$$false :- A,\tilde{A}.$$

Het redeneren met deze aangepaste hornclauses geschiedt op identieke wijze als het redeneren met normale hornclauses. Dit is mogelijk omdat we de ontkenning van een predikaat op vatten als een nieuw predikaat.

5.1 Een prototype voor ATMS*

5.1.1 Datastrukturen

De knopen van ATMS* worden gerepresenteerd met het predikaat *knoop*(I,D). Dit predikaat bezit twee argumenten, namelijk een identifikatie I en een datastruktuur D. De identifikatie gebruiken we om de verwijzingen te realiseeren. De datastruktuur kan bestaan uit drie verschillende functoren.

- De functor *afgeleid*(B,L,G,R) representeert een afgeleide knoop, waarin B de bewering, L het label, G de lijst met knopen, die met behulp van deze knoop gerechtvaardigd worden en R de lijst met rechtvaardigingen voor deze knoop representeert.
- De functor *aanname*(B,I,Ger,Gen,Kl) representeert een aanname, waarin B de bewering, I een bitvektorlijst voor de identificatie van het bit behorende bij een aanname, Ger de lijst met knopen, die met behulp van deze knoop gerechtvaardigd worden, Gen een verwijzing naar een generatieve knoop of het atoom premisse en Kl de klasse van de premisse representeert.
- De functor *generatief*(I,B,Kl) representeert een aanname, waarin de bewering B een generatieve premisse, I de lijst met verwijzingen naar de aannamen, die de instanties van de generatieve premisse representeren, en Kl de klasse van de generatieve premisse representeert.

Het label van een afgeleide knoop bestaat uit een lijst omgevingen. Elke omgeving wordt gerepresenteerd door een functor *omgeving*(K,B), waarin K de klasse van de omgeving en B de bitvektorlijst van de omgeving representeert. Omdat bitvektoren niet in prolog gerepresenteerd kunnen worden, gebruiken we hiervoor een lijst. Elk element uit de lijst kan het atoom *in* of het atoom *uit* bevatten.

De knoop met identifikatie 1 zullen we gebruiken voor de representatie van de knoop *false*.

De verzameling nogood wordt gerepresenteerd met behulp van het fact *nogood*(X), waarin X een

lijst met omgevingen representeert.

Omdat in dit prototype het verwijderen van knopen niet geimplementeerd wordt, is het niet nodig om alle vrije identifikatie nummers bij te houden. Het fact *id*(*N*) geeft het eerste vrije identifikatienummer voor een nieuwe knoop weer.

Om de zelfde reden als bij de identifikatienummers kan bij de registratie van de vrije bits uit de bitvektor volstaan worden met het bijhouden van het eerste vrije bit. Dit geschiedt door het nummer van het eerste vrije bit in de bitvektor op te slaan met het predikaat *bitnr*(*N*).

5.1.2 Interface naar het redeneersysteem

De interface tussen ATMS* en het redeneersysteem wordt gerealiseerd met behulp van de volgende prolog predikaten:

— Het predikaat *afgeleid*(*B*,*I*) plaatst voor een bewering *B* een afgeleide knoop in ATMS*. Deze knoop krijgt de identifikatie *I*.

— Het predikaat *aanname*(*B*,*I*1,*Gen*,*K*,*I*2) plaatst een aanname voor de bewering *B* in ATMS*. Deze aanname krijgt de identifikatie *I*1. Indien deze aanname een instantie is van een generatieve premisse dan moet *Gen* een verwijzing bevatten naar de bijbehorende generatieve knoop, zo niet dan moet *Gen* gelijk aan *premisse* zijn. De klasse van de aaname wordt weergegeven door *K*. Wanneer een aanname aan ATMS* wordt toegevoegd, wordt tevens een korresponderende afgeleide knoop toegevoegd indien deze nog niet aanwezig is. De identifikatie van de afgeleide knoop is *I*2.

— Het predikaat *generatief*(*B*,*I*,*K*) plaatst een generatieve knoop in ATMS* voor de bewering *B*. Deze bewering bevindt zich in de klasse *K* en krijgt een identifikatie *I*.

— Het predikaat *instantie*(*I*1,*B*,*I*2,*I*3) plaatst een nieuwe aanname in ATMS* voor de bewering *B*.. Deze aanname is een instantie van de generatieve knoop met identifikatie *I*1. De identifikatie van de toegevoegde aanname is *I*2 en de identifikatie van de korresponderende afgeleide knoop is *I*3.

— Het predikaat *rechtvaardiging*(*C*,*A*) plaatst een rechtvaardiging voor de bewering met identifikatie *C* in ATMS*. Deze bewering wordt gerechtvaardigd door de beweringen met identifikaties *A*.

Het deduktiesysteem is in het prototype gerealiseerd met behulp van het predikaat *deduktie*(*Max*). Dit predikaat realiseert alle mogelijke afleidingen met maximaal *Max* afleidingsstappen. Met behulp van het predikaat *afleidbaar*(*B*,*W*) kan vervolgens nagegaan worden of een bewering *B* afleibaar is, voor waar gehouden kan worden, in een mogelijk wereld. Het argument *W* bevat de lijst van deze mogelijke werelden. Deze werelden worden weergegeven met behulp van van bitvektoren.

Ten slotte bevat het prototype het predikaat *demo*(*In*,*Uit*,*Max*). Dit predikaat leest de premissen uit de file *In*. De premissen moeten in de volgende vorm in de invoerfile staan.

Niet generatieve premissen :	*prem*(*Bewering*,*Kl*).
Generatieve premissen :	*gen*(*Bewering*,*Kl*).

Met deze premissen realiseert het predikaat *demo* alle mogelijke afleidingen met maximaal *Max* afleidingsstappen. Vervolgens worden in de file *Uit* alle mogelijke werelden afdrukt met de beweringen die in deze werelden afleidbaar zijn.

5.1.3 Primaire funkties

De primaire funkties zijn gerealiseerd met behulp van de volgende predikaten. Sommige van deze predikaten worden voor de interface naar het redeneer systeem gebruikt. Zulke predikaten zijn ook in het vorige hoofdstuk, dat handelt over de interface naar het redeneersysteem, beschreven.

— Het predikaat *afgeleid*(*B*,*I*) plaatst voor een bewering *B* een afgeleide knoop in ATMS*. Deze knoop krijgt de identifikatie *I*.

— Het predikaat *aanname*(*B*,*I*1,*Gen*,*K*,*I*2) plaatst een aanname voor de bewering *B* in ATMS*. Deze aanname krijgt de identifikatie *I*1. Indien deze aanname een instantie is van een generatieve premisse dan moet *Gen* een verwijzing bevatten naar de bijbehorende generatieve knoop, zo niet dan moet *Gen* gelijk aan *premisse* zijn. De klasse van de aaname wordt weergegeven door *K*. Wanneer een aanname aan ATMS* wordt toegevoegd, wordt tevens een korresponderende afgeleide knoop toegevoegd indien deze nog niet aanwezig is. De identifikatie van de afgeleide knoop is *I*2.

— Het predikaat *generatief*(*B*,*I*,*K*) plaatst een generatieve knoop in ATMS* voor de bewering *B*. Deze bewering bevindt zich in de klasse *K* en krijgt een identifikatie *I*.

— Het predikaat *instantie*(*I*1,*B*,*I*2,*I*3) plaatst een nieuwe aanname in ATMS* voor de bewering *B*.. Deze aanname is een instantie van de generatieve knoop met identifikatie *I*1. De identifikatie van de toegevoegde aanname is *I*2 en de identifikatie van de korresponderende afgeleide knoop is *I*3.

— Het predikaat *rechtvaardiging*(*C*,*A*) plaatst een rechtvaardiging voor de bewering met identifikatie *C* in ATMS*. Deze bewering wordt gerechtvaardigd door de beweringen met identifikaties *A*.

— Het predikaat *werk_label_bij*(*I*,*O*) werkt het label voor de knoop met de identifikatie *I* bij met de omgevingen *O*.

— Het predikaat *ng_bijwerken* werkt de verzameling nogood bij met de omgevingen uit het label van de knoop *false*.

— Het predikaat *wereld* bepaalt met behulp van de verzameling nogood de mogelijke werelden waarin men kan geloven.

6. Konklusie

In de voor gaande hoofdstukken is een logika behandeld voor het redeneren met onzekere en onvolledige kennis. De essentie van deze logika is de samenhang tussen de geloofwaardigheid en inkonsistentie. Gebleken is dat deze logika voor eenvoudige voorbeelden een goede beschrijving geeft van het redeneren met onzekere en onvolledige kennis. Hoe deze logika zich bij komplexere problemen gedraagt moet nog nader onderzocht worden. Andere punten voor nader onderzoek zijn de betekenis van de implikatie [zie hoofdstuk 2.7], heuristieken voor het redeneersysteem en het plannen van akties bij een toepassing voor robot besturingen.

7. Literatuurlijst

1. Bibel, W.
   A deductive solution for plan generation
   New Generation Computing **4** (1986) 115-132
2. Bossu, G. , Siegel, P.
   Saturation, Nonmonotonic reasoning and closed world assumption.
   Artificial Intelligence **25** (1985) 13-63
3. Coelo, H. , Cotta, J. C. , Pereira, L. M.
   How to solve it with prolog
   Laboratorio National de Engenharia Civil, Lisbou, Protugal (1980)
4. Davis, M.
   The mathematics of nonmotonic reasoning.
   Artificial Intelligence **13** (1980) 73-80
5. Davis, R. H. , Comacho M.
   The application of logic programming to the generation of plans of robots
   Robotica **2** (1984) 137-146
6. Doyle, J.
   A truth maitenance system.
   Artificial Intelligence **12** (1979) 231-272
7. Gabbay, D.
   Intuitionistic basis for nonomonotonic logic.
   Procedings Conference on Automated Deduction (1982) 260-273
8. de Kleer, J.
   An assumption based TMS.
   Artificial Intelligence **28** (1986) 127-162
9. de Kleer, J.
   Extending the ATMS.
   Artificial Intelligence **28** (1986) 163-196
10. Lifschitz, V.
    Closed world databases and circumcription.
    Artificial Intelligence **27** (1985) 229-235
11. Lukaszewicz, W.
    Nonmonotonic logic for default theories
    ECCAI (1985) 403-412
12. Mc Carthy, J.
    Circumscription: a form of nonmonotonic reasoning.
    Artificial Intelligence **13** (1980) 27-39
13. Mc Carty, J.
    Addendum: circumsciption and other nonmonotonic formalisms
    Artificial Intelligence (1980) 171-172
14. Mc Dermott, D. , Doyle, J.
    Nonmonotonic reasoning I.
    Artificial Intelligence **13** (1980) 42-72
15. Mc Dermott, D.
    Nonomontonic logic II: Nonmonotonic modal models.
    Journal of the Association for Computing Machinery **29** (1982) 33-57
16. Mendelson, E.
    Introduction to Mathematical Logic

Van Nostrand-Reinhold, New York (1964)

17. Moore, R.C.
Semantical considerations on nonomonotonic logic.
Artificial Intelligence **25** (1985) 75-94

18. Reiter, R.
On closed world databases.
In : Gallaire, H. Minker, J.
Logic and databases.
Plenum Press, New York (1978)

19. Reiter, R.
A logic for default reasoning
Artificial Intelligence **13** (1980) 81-132

20. Reiter, R. , Criscuolog, G.
On interacting defaults
Proc. of the 7th IJCAI (1981) 270-276

21. Rescher, N.
Hypothetical Reasoning
North-Holland Publishing Company, Amsterdam (1964)

22. Roos, N.
Geloofwaardig argumenteren, kwalitatief beschouwd.
TU Delft (1986)

23. Rosenschein, S. J.
Plan synthesis: a logical perspective
Proc. of the 7th IJCAI (1981) 331-337

24. Turner, R.
Logic for AI.
Ellis Horwood (1984)

8. Bijlage I : listings van het prototype

8.1 data

```
id(2). /* Het eerste vrije identifikatie nummer */

knoop(1,afgeleid(false,[],[],[])). /* De knoop false */

nogood([]). /* De verzameling nogood */

bitnr(1). /* Het eerste vrije bit in de bitvektor */
```

8.2 hulpfunk1

```
/* Het predikaat verenig(BitV1,BitV2,ResultV) bepaalt de vereniging van twee bit vektoren. */

verenig([],X,X).

verenig(X,[],X).

verenig([in|X],[_|Y],[in|Z]) :-
        verenig(X,Y,Z),!.

verenig([_|X],[in|Y],[in|Z]) :-
        verenig(X,Y,Z),!.

verenig([uit|X],[uit|Y],[uit|Z]) :-
        verenig(X,Y,Z).

/* Het predikaat deelverz(BitV1,BitV2) gaat na of de bitvektor BitV1 een deelverzameling is van
BitV2 */

deelverz([],_).

deelverz(X,[]) :-
        is_uit(X).

deelverz([in|X],[in|Y]) :-
        deelverz(X,Y).

deelverz([uit|X],[_|Y]) :-
        deelverz(X,Y).

/* Het predikaat is_uit(BitV) gaat na of alle elementen uit de bitvektor BitV de waarde uit
hebben. */

is_uit([]).

is_uit([uit|X]) :-
        is_uit(X).

/* Het predikaat member(X,Y) gaat na of het element X voor komt in de lijst Y */

member(X,[X|_]).

member(X,[Y|Z]) :-
        member(X,Z).
```

```
/* Het predikaat append(X,Y,Z) maakt van de lijsten X en Y een nieuwe lijst Z. */

append([],X,X).

append([X|Y],Z,[X|Q]) :-
      append(Y,Z,Q).

/* Het predikaat minimum(X,Y,Z) bepaalt het minimum van de twee getallen X en Y. Het
resultaat wordt in Z geplaatst. */

minimum(X,Y,X) :-
      X < Y.

minimum(X,Y,Y) :-
      X >= Y.

/* Het predikaat sorteer(X,Y,N) is waar als lijst Y de gesorteerde lijst X is. Indien N=0 dan wordt
er gesoorteerd op de waarde van de elementen. Indien N>0 dan dan wordt er gesorteerd op de
waarde van het N-de argument van een element. */

sorteer([],[],N).

sorteer([X|Y],Z,N) :-
      splits(X,Y,P,Q,N),
      sorteer(P,U,N),
      sorteer(Q,V,N),
      append(U,[X|V],Z).

splits(_,[],[],[],N).

splits(X,[Y|Z],[Y|P],Q,0) :-
      Y<X,
      splits(X,Z,P,Q,0).

splits(X,[Y|Z],P,[Y|Q],0) :-
      Y>=X,
      splits(X,Z,P,Q,0).

splits(X,[Y|Z],[Y|P],Q,N) :-
      arg(N,X,U),
      arg(N,Y,V),
      V<U,
      splits(X,Z,P,Q,N).

splits(X,[Y|Z],P,[Y|Q],N) :-
      arg(N,X,U),
      arg(N,Y,V),
      V>=U,
      splits(X,Z,P,Q,N).

/* Het predikaat reverse(X,Y) is als lijst Y gelijk is aan lijst X in om gekeerde volgorde. */

reverse(X,Y) :-
      rev(X,[],Y).

rev([],X,X).
```

```
rev([P|Q],R,S) :-
      rev(Q,[P|R],S).
```

8.3 hulpfunk2

/* Het predikaat minimaal(X,Y,Z) verwijdert uit de lijst X die omgevingen die niet minimaal zijn t.o.v. de omgevingen uit de lijst Y. D.w.z. dat de omgevingen uit de lijst X, die een superverzameling zijn van een omgeving uit de lijst Y, verwijderd worden uit X. Het resultaat hiervan staat in Z. */

```
minimaal([],_,[]).

minimaal([Omgeving|Rest],X,[Omgeving|Nrest]) :-
      kontroleer(Omgeving,X),
      minimaal(Rest,X,Nrest).

minimaal([_|Rest],X,Nrest) :-
     minimaal(Rest,X,Nrest).
```

/* Het predikaat kontroleer(X,Y) is waar als omgeving X geen super verzameling is van een omgeving uit de lijst Y. */

```
kontroleer(_,[]) :- !.

kontroleer(omgeving(_,BitV1),[omgeving(_,BitV2)|Rest]) :-
      not(deelverz(BitV2,BitV1)),
      kontroleer(omgeving(_,BitV1),Rest), !.
```

/* Het predikaat minimaliseer(Omgev1,Omgev2) verwijdert uit de lijst met omgevingen Omgev1, die omgevingen die een superverzameling zijn van een andere omgeving. Het resultaat wordt in Omgev2 geplaatst. */

```
minimaliseer([],[]).

minimaliseer([Omgeving|Omgevingen],Resultaat) :-
      minimaliseer(Omgevingen,HulpResul),
      ( kontroleer(Omgeving,HulpResul),
        Resultaat = [Omgeving|HulpResul]
      ; Resultaat = HulpResul
      ).
```

/* Het predikaat eq(X,Y) is waar als X en Y de zelfde struktuur hebben. */

```
eq(X,Y) :-
      var(X),
      var(Y).

eq(X,Y) :-
      nonvar(X),
      nonvar(Y),
      functor(X,F,N),
      functor(Y,F,N),
      eqarg(X,Y,N).

eqarg(_,_,0).

eqarg(X,Y,N) :-
```

```
        arg(N,X,Arg1),
        arg(N,Y,Arg2),
        eq(Arg1,Arg2),
        M is N-1,
        eqarg(X,Y,M).

/* Het predikaat instantie_van(X,Y) is waar als bewering X een instanties van bewering Y is. */

instantie_van(X,Y) :-
        var(Y).

instantie_van(X,Y) :-
        nonvar(X),
        nonvar(Y),
        functor(X,F,N),
        functor(Y,F,N),
        instantie_arg(X,Y,N).

instantie_arg(_,_,0).

instantie_arg(X,Y,N) :-
        arg(N,X,Arg1),
        arg(N,Y,Arg2),
        instantie_van(Arg1,Arg2),
        M is N-1,
        instantie_arg(X,Y,M).
```

8.4 primairefunk1

```
/* De verzameling nogood wordt bijgewerkt na het aanroepen van het predikaat ng_bijwerken */

ng_bijwerken :-
        knoop(1,afgeleid(false,[],_,_)). /* Een leeg label */

ng_bijwerken :-
        retract( knoop(1,afgeleid(false,Label,_,Rechtv)) ),
        retract(nogood(Oudedata)),
        verwerk_omg(Label,Oudedata,Nieuwedata),
        asserta( nogood(Nieuwedata) ),
        asserta( knoop(1,afgeleid(false,[],_,Rechtv)) ),
        knopen_konsistent(Label,2).

/* Het predikaat verwerk_omg(Label,Oudedata,Nieuwedata) maakt van de verzamelingen
minimale omgevingen Label en Oudedata een nieuwe verzameling minimale omgevingen
Nieuwedata. */

verwerk_omg(Label,Oudedata,Nieuwedata) :-
        minimaal(Label,Oudedata,Hulpdata1),
        minimaal(Oudedata,Hulpdata1,Hulpdata2),
        append(Hulpdata1,Hulpdata2,Nieuwedata).

/* Het predikaat knopen_konsistent(Label,N) is waar als alle knopen met een identifikatie
nummer groter dan of gelijk aan N een label hebben, waarin geen inkonsistente omgevingen
voorkomen. */

knopen_konsistent(_,N) :-
        id(N).
```

```
knopen_konsistent(Label1,N) :-
      not( id(N) ),
      retract(knoop(N,afgeleid(Bew,Label2,Ger,Recht)) ),
      minimaal(Label2,Label1,Nieuw_1),
      asserta(knoop(N,afgeleid(Bew,Nieuw_1,Ger,Recht)) ),
      M is N+1,
      knopen_konsistent(Label1,M).

knopen_konsistent(Label,N) :-
      M is N+1,
      knopen_konsistent(Label,M).
```

8.5 primairefunk2

```
/* Het predikaat rechtvaardiging(Antecedenten,Consequent) voegt een rechtvaardiging toe voor de
knoop met identificatie Consequent */

rechtvaardiging(Antecedenten,Consequent) :-
      sorteer(Antecedenten,Santeced,0),
      knoop(Consequent,afgeleid(_,_,_,Recht)),
      ( member(Santeced,Recht)
      ; retract(knoop(Consequent,afgeleid(Bew,Label,Ger,Recht))),
        asserta(knoop(Consequent,afgeleid(Bew,Label,Ger,
            [Santeced|Recht]))),
        verwerk_anteced(Santeced,Consequent),
        ( leeg_label(Santeced)
        ; setof(Omgeving,bepaal(Santeced,Omgeving),
            Omgevingen),
          werk_label_bij(Consequent,Omgevingen) ) ), !.

/* Het predikaat verwerk_anteced(Antecedenten,Consequent) is waar als bij elke antecedent uit de
lijst Atecedenten een verwijzing naar de consequent knoop Consequent geplaatst is */

verwerk_anteced([],_).

verwerk_anteced([Ident|Rest],Consequent):-
    ( ( knoop(Ident,afgeleid(_,_,Ger,_)),
        member(Consequent,Ger) )
    ; retract(knoop(Ident,afgeleid(Bew,Label,Ger,Recht))),
      asserta(knoop(Ident,afgeleid(Bew,Label,[Consequent|Ger],
            Recht)))
    ),
    verwerk_anteced(Rest,Consequent).

/* Het predikaat bepaal(Antecedenten,Omgeving) is waar als een omgeving Omgeving voor de
consequent bepaald is. Deze omgeving is afhankelijk van de antecedenten Antecedenten van een
rechtvaardiging. */

bepaal([],omgeving(1,[])).

bepaal([Ident|Rest],Omgeving) :-
      bepaal(Rest,Hulp_omg1),
      knoop(Ident,afgeleid(_,Label,_,_)),
      member(Hulp_omg2,Label),
      Hulp_omg1 = omgeving(Klasse1,BitV1),
      Hulp_omg2 = omgeving(Klasse2,BitV2),
      verenig(BitV1,BitV2,BitV3),
```

```
        minimum(Klasse1,Klasse2,Klasse3),
        Omgeving = omgeving(Klasse3,BitV3).

/* Het predikaat leeg_label(IdentLijst) is waar als er een knoop is met een identifikatie die
voorkomt in de IdentLijst waarvan het label leeg is. */

leeg_label([Ident|_]) :-
        knoop(Ident,afgeleid(_,[],_,_)).

leeg_label([_|Rest]) :-
        leeg_label(Rest).
```

8.6 primairefunk3

```
/* Het predikaat generatief(Bew,Ident,Klasse) plaatst een generative knoop voor de premisse Bew
in de database. Deze premisse bevind zich in de klasse Klasse. Deze knoop krijgt de identifikatie
Ident */

generatief(Bew,Ident,Klasse) :-
        retract(id(Ident)),
        N is Ident + 1,
        asserta(id(N)),
        asserta(knoop(Ident,generatief([],Bew,Klasse)) ).

/* Het predikaat afgeleid(Bew,Ident) plaatst ee afgeleide knoop voor de bewering Bew in de
database. Deze knoop krijgt de identifikatie Ident */

afgeleid(Bew,Ident) :-
        retract(id(Ident)),
        N is Ident + 1,
        asserta(id(N)),
        asserta(knoop(Ident,afgeleid(Bew,[],[],[]))),
        verwerk_instantie(Ident).

/* Het predikaat aanname(Bew,Ident1,Gen,Klasse,Ident2) plaatst een aanname voor de bewering
Bew in de database. Deze bewering krijgt een identifikatie Ident1 en heeftbevind zich in de
geloofwaardigheids klasse Klasse. Indien de aanname een instantie is van een generative premisse
dan bevat het argument Gen een verwijzing naar de generative knoop. De bij deze aanname
behorende afgeleide knoop heeft een identifikatie Ident2. */

aanname(Bew,Ident1,Gen,Klasse,Ident2) :-
        retract(id(Ident1)),
        N is Ident1 + 1,
        asserta(id(N)),
        retract(bitnr(Bitnr)),
        M is Bitnr + 1,
        asserta(bitnr(M)),
        maak_bitvek(Bitnr,BitVek),
        ( knoop(Ident2,afgeleid(Bew2,_,_,_)),
          eq(Bew,Bew2)
        ; afgeleid(Bew,Ident2)
        ),
        asserta(knoop(Ident1,aanname(Bew,BitVek,Ident2,
              Gen,Klasse))),
        retract(knoop(Ident2,afgeleid(Bew,Label,Ger,Recht))),
        asserta(knoop(Ident2,afgeleid(Bew,Label,Ger,[[Ident1]|Recht]))),
        Nieuwe_omg = omgeving(Klasse,BitVek),
```

```
    werk_label_bij(Ident2,[Nieuwe_omg]).

/* Het predikaat maak_bitvek(Bitnr,BitVek) is waar als het bit Bitnr van de bitvektor BitVek de
waarde in heeft en de overige bits de waarde uit. */

maak_bitvek(1,[in]).

maak_bitvek(N,[uit|X]) :-
    M is N-1,
    maak_bitvek(M,X).

/* Het predikaaat instantie(Ident1,Bew,Ident2,Ident3) voegt een aanname voor de generative
premisse met identifikatie Ident1 toe. De instantie is de bewering Bew. De nieuwe aanname heeft
de identifikatie Ident2 en de bijbehorende afgeleide knoop de identifikatie Ident3 */

instantie(Ident1,Bew,Ident2,Ident3) :-
    knoop(Ident1,generatief(_,_,Klasse)),
    aanname(Bew,Ident2,Ident1,Klasse,Ident3),
    retract(knoop(Ident1,generatief(Inst,Prem,Klasse))),
    asserta(knoop(Ident1,generatief([Ident2|Inst],Prem,Klasse))).
```

8.7 primairefunk4

```
/* Het predikaat werk_label_bij(Ident,Omgev) werkt het label voor de knoop met identifikatie
Ident bij met de nieuwe omgevingen Omgev voor dit label. */

werk_label_bij(Ident,Omgev) :-
    minimaliseer(Omgev,Omgev1),
    nogood(NG),
    minimaal(Omgev1,NG,Omgev2),
    knoop(Ident,afgeleid(_,Label,_,_)),
    minimaal(Omgev2,Label,Omgev3),
    ( Omgev3 = []
    ; minimaal(Label,Omgev3,Omgev4),
      append(Omgev3,Omgev4,NieuwLabel),
      retract(knoop(Ident,afgeleid(Bewering,_,Gerechtv,Rechtvaard))),
      asserta(knoop(Ident,afgeleid(Bewering,NieuwLabel,Gerechtv,
          Rechtvaard))),
      ( Gerechtv = []
      ; nieuwe_omg(Omgev3,Ident,Gerechtv)
      ),
      ( not( Bewering = false )
      ; ng_bijwerken
      )
    ).

/* Het predikaat nieuwe_omg(Omgev,Ident,Gerechtv) bepaalt de nieuwe omgevingen voor de
knopen uit de lijst van knopen Gerechtv, die m.b.v. de knoop Ident gerechtvaardigd worden.
Hierbij zijn alleen de nieuwe omgevingen Omgev van de knoop Ident van belang */

nieuwe_omg(Omgevingen,Ident,[]).

nieuwe_omg(Omgevingen,Ident1,[Ident2|Rest]) :-
    setof(Omgeving,bepaal_omg(Ident1,Ident2,
        Omgevingen,Omgeving),LijstOmgev),
    werk_label_bij(Ident2,LijstOmgev),
    nieuwe_omg(Omgevingen,Ident1,Rest).
```

```
/* Het predikaat bepaal_omg(Ident1,Ident2,Omgevingen,Omgeving) bepaalt een omgeving
Omgeving voor een rechtvaardiging van de knoop met identifikatie Ident2, die de knoop met
identifikatie Ident1 bevat. Aan het label van de knoop met identifikatie Ident1 zijn de omgevingen
Omgevingen toegevoegd. */

bepaal_omg(Ident1,Ident2,Omgevingen,Omgeving) :-
        knoop(Ident2,afgeleid(_,_,_,Rechtv)),
        member(Een_Rechtv,Rechtv),
        member(Ident1,Een_Rechtv),
        konstr_omg(Een_Rechtv,Ident1,Omgevingen,Omgeving).

/* Het predikaat konstr_omg(Een_rechtv,Ident,Omgevingen,Omgeving) konstrueerd een omgeving
voor de huidige knoop m.b.v. de rechtvaardiging Een_rechtv. Bij de knoop met identifikatie Ident
zijn alleen de nieuwe omgevingen Omgevingen van belang. */

konstr_omg([],_,_,omgeving(0,[])).

konstr_omg([Ident|Rest],Ident,Omgevingen,Omgeving) :-
        konstr_omg(Rest,Ident,Omgevingen,Hulp_omg1),
        Hulp_omg1 = omgeving(Klasse1,Bit_vek1),
        member(Hulp_omg2,Omgevingen),
        Hulp_omg2 = omgeving(Klasse2,Bit_vek2),
        verenig(Bit_vek1,Bit_vek2,Nieuwe_BV),
        minimum(Klasse1,Klasse2,Nieuwe_K1),
        Omgeving = omgeving(Nieuwe_K1,Nieuwe_BV).

konstr_omg([Ident2|Rest],Ident1,Omgevingen,Omgeving) :-
        not( Ident1 = Ident2 ),
        konstr_omg(Rest,Ident1,Omgevingen,Hulp_omg1),
        Hulp_omg1 = omgeving(Klasse1,Bit_vek1),
        knoop(Ident2,afgeleid(_,Label,_,_)),
        member(Hulp_omg2,Label),
        Hulp_omg2 = omgeving(Klasse2,Bit_vek2),
        verenig(Bit_vek1,Bit_vek2,Nieuwe_BV),
        minimum(Klasse1,Klasse2,Nieuwe_k1),
        Omgeving = omgeving(Nieuwe_k1,Nieuwe_BV).
```

8.8 primairefunk5

```
/* Het predikaat wereld bepaald m.b.v. de verzameling nogood wat de mogelijke werelden zijn,
waarin men kan geloven. */

wereld :-
        setof(K1,een_klasse(K1),KlasseLijst),
        setof(Wereld,een_wereld(KlasseLijst,Wereld),Werelden1),
        minimaliseer(Werelden1,Werelden2),
        asserta(werelden(Werelden2)).

/* Het predikaat een_wereld(K1_Lijst,Wereld) is waar als de wereld Wereld een wereld is voor de
klasse uit lijst K1_Lijst. */

een_wereld([],omgeving(_,[])).

een_wereld([K1|Rest],Wereld) :-
        een_wereld(Rest,Hulp_wereld),
        voeg_klassen_toe(K1,Hulp_wereld,Wereld).
```

```
/* Het predikaat voeg_klassen_toe(N,Wereld1,Wereld2) is waar als een maximaal aantal
premissen uit de klasse N aan wereld Wereld1 zijn toegevoegd. Het resultaat hiervan staat in
Wereld2. */

voeg_klassen_toe(N,omgeving(_,BitV1),Wereld2) :-
        assertz(truuk),                 /* een truuk    */
      knoop(_,aanname(_,BitV3,_,_,N)),
      not(deelverz(BitV3,BitV1)),
      verenig(BitV3,BitV1,BitV4),
      nogood(Nogood),
      kontroleer(omgeving(_,BitV4),Nogood),
       ( not(truuk)
       ; retract(truuk) ),              /* een truuk    */
      voeg_klassen_toe(N,omgeving(_,BitV4),Wereld2).

voeg_klassen_toe(N,Wereld,Wereld) :-
        retract(truuk).                 /* een truuk    */

/* Het predikaat een_klasse(Kl) is waar als Kl de waarde heeft van een in de premisse gebruikte
klasse heeft. */

een_klasse(Kl) :-
      knoop(_,aanname(_,_,_,_,Kl)).
```

8.9 redeneer1

```
:- op(1,fx,~).

/* Het predikaat deduktie(Max) konstrueert alle afleidingen met een aantal afleidingsstappen
kleiner dan of gelijk aan Max */

deduktie(Max) :-
      ( setof(Rule,een_rule(Rule),Rulelijst)
      ; Rulelijst = []
      ),
      afleiding(Max,Rulelijst),
      setof(Recht,inkonsistent(Recht),RechtLijst),
      verwerk_ink(RechtLijst),
      wereld.

/* Het predikaat voetoe(Prem) plaatst voor elk elementen uit de file met premissen Prem een
knoop in de database. */

voegtoe(Prem) :-
      see(Prem),
      lees_data,
      seen.

/* Het predikaat lees_data lees elementen uit de file en plaatst voor elk element een knoop in de
database. */

lees_data :-
      read(In),
      ( In = end_of_file
      ; In = gen(Bew,Klasse),
        tab(4),
        write(In),nl,
```

```
        generatief(Bew,_,Klasse),
        lees_data
    ; In = prem(Bew,Klasse),
        tab(4),
        write(In),nl,
        aanname(Bew,_,premisse,Klasse,_),
        lees_data
    ).

/* Het predikaat afleiding(Max) realiseert alle afleidingen met maximaal Max afleidingsstappen. */

afleiding(_,[]).

afleiding(0,_).

afleiding(N,Rulelijst) :-
    M is N - 1,
    id(Y),
    verwerk_rules(Rulelijst,Y),
    ( id(Z),
      Z=Y
    ; afleiding(M,Rulelijst) ).

/* Het predikaat verwerk_rules(Rulelijst,Y) probeert elke rule uit de rulelijst Rulelijst toe te
passen. Hierbij mogen geen knopen gebruikt worden met een identifikatie groter dan of gelijk aa Y.
*/

verwerk_rules([rule(Ident,(Head :- Body),Type)|Rest],X) :-
    satisfy(Body,X,Recht),
    verwerk_result(Recht,rule(Ident,(Head :- Body),Type)),
    fail.

verwerk_rules([_|Rest],X) :-
    verwerk_rules(Rest,X).

verwerk_rules([],_).

/* Het predikaat verwerk_result(Recht,Rule) is waar als de rule Rule verwerkt is. De consequent
van deze rule krijgt een rechtvaardiging, die samengesteld is uit de lijst Recht met daar aan
toegevoegd de indentifikatie van de rule. */

verwerk_result(Recht,rule(Ident1,(Head :- Body),generatief)) :-
    ( knoop(_,aanname(Bew1,_,Ident3,Ident1,_)),
      eq(Bew1,(Head :- Body))
    ; instantie(Ident1,(Head :- Body),_,Ident3)
    ), !,
    ( knoop(Ident2,afgeleid(Bew2,_,_,_)),
      eq(Head,Bew2),
      rechtvaardiging([Ident3|Recht],Ident2)
    ; afgeleid(Head,Ident2),
      rechtvaardiging([Ident3|Recht],Ident2)
    ),
    verwerk_instantie(Ident2), !.

verwerk_result(Recht,rule(Ident1,(Head :- Body),premisse)) :-
    ( knoop(Ident2,afgeleid(Bew,_,_,_)),
```

```
        eq(Head,Bew),
        rechtvaardiging([Ident1|Recht],Ident2)
    ;   afgeleid(Head,Ident2),
        rechtvaardiging([Ident1|Recht],Ident2)
    ),
    verwerk_instantie(Ident2), !.
```

/* Het predikaat satisfy(Body,X,Recht) probeert feiten voor de elementen uit de body Body van de rule te vinden. De identifikatie van deze feiten moeten kleiner dan X zijn. De identifikaties van de gebruikte feiten worden in de lijst Recht geplaatst. Hiermee kan dan een rechtvaazing gekonstrueerd worden. */

```
satisfy((Feit,Rest),X,[Ident|Recht]) :-
    knoop(Ident,afgeleid(Feit,_,_,_)),
    Ident < X,
    satisfy(Rest,X,Recht).

satisfy(Feit,X,[Ident]) :-
    knoop(Ident,afgeleid(Feit,_,_,_)),
    Ident < X.
```

/* Het predikaat inkonsistent(Recht) probeert een tegenspraken af te leiden. De idetifikaties behorende bij de tegenspraak worden in de lijst Recht geplaatst. */

```
inkonsistent([Ident1,Ident2]) :-
    knoop(Ident1,afgeleid(~Pred,_,_,_)),
    knoop(Ident2,afgeleid(Pred,_,_,_)).
```

/* Het predikaat verwerk_ink(RechtLijst) plaatst rechtvaardigingen voor de knoop false in de database. RechtLijst bevat de rechtvaardigingen voor de knoop false. */

```
verwerk_ink([]).

verwerk_ink([Recht|Rest]) :-
    rechtvaardiging(Recht,1),
    verwerk_ink(Rest).
```

/* Het predikaat een_rule(Rule) is waar als er een knoop is voor een geneativepremisse of afgeleidebewering, wiens bewering een rule is. */

```
een_rule(Rule) :-
    knoop(Ident,afgeleid(Bew,_,_,_)),
    functor(Bew,(:-),2),
    Rule = rule(Ident,Bew,premisse).

een_rule(Rule) :-
    knoop(Ident,generatief(_,Bew,_)),
    functor(Bew,(:-),2),
    Rule = rule(Ident,Bew,generatief).
```

/* Het predikaat verwerk_instantie(Ident) legt relaties tussen de bewering met identifikatie Ident en instanties van deze bewering. Ook worden relaties gelegd tussen de beweringen, waarvan de bewering met identifikatie Ident een instantie is, en die bewering. */

```
verwerk_instantie(Ident1) :-
    knoop(Ident1,afgeleid(Bew1,_,_,_)),
```

```
        knoop(Ident2,afgeleid(Bew2,_,_,_)),
        ( instantie_van(Bew1,Bew2),
          not( eq(Bew1,Bew2) ),
          rechtvaardiging([Ident2],Ident1)
        ; instantie_van(Bew2,Bew1),
          not( eq(Bew1,Bew2) ),
          rechtvaardiging([Ident1],Ident2)
        ),
        fail.

verwerk_instantie(_).
```

8.10 redeneer2

```
/* het predikaat afleidbaar(Bew,Werelden) is waar als het arument Werelden alle werelden bevat
waarin de bewering Bew afgeleid is. De wereld bevat ook de klasse van de bewering in die wereld.
*/

afleidbaar(Bew,Werelden1) :-
        knoop(_,afgeleid(Bew,Label,_,_)),
        setof(Wereld,zoek_op(Label,Wereld),Werelden2),
        verwijder_mv(Werelden2,Werelden1).

/* Het predikaat zoek_op(Label,Wereld) is waar als een omgeving uit het label Label binnen een
wereld Wereld valt. Bij de wereld wordt de geloofwaardigheid van de bewering in een bepaalde
omgeving geregistreerd. */

zoek_op(Label,Wereld_Klasse) :-
        member(Omgeving,Label),
        werelden(Werelden),
        member(Wereld,Werelden),
        Omgeving = omgeving(Klasse,BitV1),
        Wereld = omgeving(_,BitV2),
        deelverz(BitV1,BitV2),
        Wereld_Klasse = omgeving(Klasse,BitV2).

/* Het predikaat konv_werelden(Werelden1,Werelden2) is als de bitvektoren van Werelden1
vervangen zijn door de verzamelingen premissen van Werelden2. */

konv_werelden([],[]).

konv_werelden([omgeving(Klasse,BitV)|Rest1], [wereld(Klasse,Prem)|Rest2]) :-
        konverteer(BitV,Prem),
        konv_werelden(Rest1,Rest2), !.

/* Het predikaat konverteer(BitV,Prem) is waar als Prem de premissen bevat, die de bitvektor
BitV representeert. */

konverteer(BitV,Prem) :-
        setof(Een_prem,bepaal_prem(BitV,Een_prem),Prem).

/* Het predikaat bepaal_prem(BitV,Een_prem) is waar als Een_prem een premisse is, die met de
bitvektor BitV gerepresenteerd wordt. */

bepaal_prem(BitV1,Prem) :-
        ( knoop(_,aanname(Prem,BitV2,_,premisse,_)),
          deelverz(BitV2,BitV1)
```

```
      ; knoop(_,generatief(_,Prem,_)) ).
```

/* Het predikaat verwijder_mv(Wer1,Wer2) verwijdert alle meervoudige voorkomens van werelden uit de verzameling Wer1. Het resultaat komt in Wer2 te recht. Bij gelijke werelden kunnen verschillende geloofwaardigheden voorkomen, afhankelijk van de gebruikte premissen. De wereld met de kleinste geloofwaardigheid wordt verwijderd. */

```
verwijder_mv(Wer1,Wer2) :-
      sorteer(Wer1,Wer3,1),
      minimaliseer(Wer3,Wer2), !.
```

8.11 demo

/* Het predikaat demo(Invoer,Uitvoer,Max) maakt alle mogelijke afleidingen met de permissen uit de invoerfile Invoer, waarbij het aantal afleidings stappen kleiner dan of gelijk aan Max is. Het resultaat hiervan wordt afgedrukt in de file Uitver. */

```
demo(Invoer,Uitvoer,Max) :-
      ( not( werelden(_) ) ; retract(werelden(_)) ),
      reconsult(data),
      tell(Uitvoer),
      printstr("De premissen :"),nl,nl,
      voegtoe(Invoer), nl,nl,
      deduktie(Max),
      werelden(Werelden),
      verwerk_wereld(Werelden),
      told.
```

/* Het predikaat resultaat drukt alle afgeleide beweringen af. */

```
resultaat(BV) :-
      afleidbaar(B,W),
      member(omgeving(Klasse,BV),W),
      printstr("Bewering : "),
      write(B),nl,
      printstr("met geloofwaardigheid : "),
      write(Klasse),nl,nl,
      fail.

resultaat(_).
```

/* Het predikaat drukaf drukt een lijst af. */

```
drukaf([]).

drukaf([X|Y]) :-
      tab(4),
      write(X),nl,
      drukaf(Y).
```

/* Het predikaat printstr(String) drukt een string af. */

```
printstr([]).

printstr([X|Y]) :-
      put(X),
      printstr(Y).
```

```
/* Het predikaat verwerk_wereld(Werelden) durkt de werelden Werelden af met de beweringen
die daarin afleidbaar zijn. */

verwerk_wereld([]).

verwerk_wereld([omgeving(_,BV)|Rest]) :-
        konverteer(BV,Prem),
        printstr("In de wereld :"),nl,nl,
        drukaf(Prem),nl,
        printstr("zijn de volgende beweringen afleidbaar :"),nl,nl,
        resultaat(BV),nl,nl,nl,nl,
        verwerk_wereld(Rest).
```

9. Bijlage II : test resultaten

1. De premissen :

```
prem(student(hans),1)
gen((volwassen(X):-student(X)),0.900000)
gen((heeft_werk(X):-volwassen(X)),0.650000)
gen((~heeft_werk(X):-student(X)),0.800000)
```

In de wereld :

```
student(hans)
heeft_werk(X):-volwassen(X)
volwassen(X):-student(X)
~heeft_werk(X):-student(X)
```

zijn de volgende beweringen afleidbaar :

Bewering : ~heeft_werk(hans)
met geloofwaardigheid : 0.800000

Bewering : ~heeft_werk(hans):-student(hans)
met geloofwaardigheid : 0.800000

Bewering : volwassen(hans)
met geloofwaardigheid : 0.900000

Bewering : volwassen(hans):-student(hans)
met geloofwaardigheid : 0.900000

Bewering : student(hans)
met geloofwaardigheid : 1

2. De premissen :

```
prem(student(hans),1)
gen((volwassen(X):-student(X)),0.900000)
gen((heeft_werk(X):-volwassen(X)),0.650000)
gen((~heeft_werk(X):-student(X)),0.800000)
prem(volwassen(hans),1)
```

In de wereld :

```
student(hans)
volwassen(hans)
heeft_werk(X):-volwassen(X)
volwassen(X):-student(X)
~heeft_werk(X):-student(X)
```

zijn de volgende beweringen afleidbaar :

Bewering : ~heeft_werk(hans)
met geloofwaardigheid : 0.800000

Bewering : ~heeft_werk(hans):-student(hans)

met geloofwaardigheid : 0.800000

Bewering : volwassen(hans):-student(hans)
met geloofwaardigheid : 0.900000

Bewering : volwassen(hans)
met geloofwaardigheid : 1

Bewering : student(hans)
met geloofwaardigheid : 1

3. De premissen :

```
prem(student(hans),1)
gen((volwassen(X):-student(X)),0.900000)
gen((heeft_werk(X):-volwassen(X)),0.650000)
gen((~heeft_werk(X):-student(X)),0.800000)
prem(volwassen(hans),1)
prem(heeft_werk(hans),1)
```

In de wereld :

```
heeft_werk(hans)
student(hans)
volwassen(hans)
heeft_werk(X):-volwassen(X)
volwassen(X):-student(X)
~heeft_werk(X):-student(X)
```

zijn de volgende beweringen afleidbaar :

Bewering : heeft_werk(hans):-volwassen(hans)
met geloofwaardigheid : 0.650000

Bewering : volwassen(hans):-student(hans)
met geloofwaardigheid : 0.900000

Bewering : heeft_werk(hans)
met geloofwaardigheid : 1

Bewering : volwassen(hans)
met geloofwaardigheid : 1

Bewering : student(hans)
met geloofwaardigheid : 1

4. De premissen :

```
prem(student(hans),1)
gen((heeft_werk(X):-volwassen(X)),0.650000)
gen((volwassen(X):-student(X)),0.900000)
gen((~heeft_werk(X):-student(X)),0.800000)
prem(student(nico),1)
prem(heeft_werk(nico),1)
prem(volwassen(cees),0.920000)
```

In de wereld :

```
heeft_werk(nico)
student(hans)
student(nico)
volwassen(cees)
heeft_werk(X):-volwassen(X)
volwassen(X):-student(X)
~heeft_werk(X):-student(X)
```

zijn de volgende beweringen afleidbaar :

Bewering : heeft_werk(nico):-volwassen(nico)
met geloofwaardigheid : 0.650000

Bewering : ~heeft_werk(hans)
met geloofwaardigheid : 0.800000

Bewering : ~heeft_werk(hans):-student(hans)
met geloofwaardigheid : 0.800000

Bewering : volwassen(hans)
met geloofwaardigheid : 0.900000

Bewering : volwassen(hans):-student(hans)
met geloofwaardigheid : 0.900000

Bewering : volwassen(nico)
met geloofwaardigheid : 0.900000

Bewering : volwassen(nico):-student(nico)
met geloofwaardigheid : 0.900000

Bewering : heeft_werk(cees)
met geloofwaardigheid : 0.650000

Bewering : heeft_werk(cees):-volwassen(cees)
met geloofwaardigheid : 0.650000

Bewering : volwassen(cees)
met geloofwaardigheid : 0.920000

Bewering : heeft_werk(nico)
met geloofwaardigheid : 1

Bewering : student(nico)
met geloofwaardigheid : 1

Bewering : student(hans)
met geloofwaardigheid : 1