Prof. Dr. Ir. R. Maes

Beslissingstabellen in kritisch perspectief

1. Inleiding

Beslissingstabellen kwamen gehandicapt ter wereld: bedoeld als een alternatief voor het klassieke programma-stroomschema, konden ze slechts heel ten dele de gerezen verwachtingen inlossen. Als algemene programma-representatietechniek werden ze dan ook, terecht, overschaduwd en vervangen door Nassi-Shneiderman-diagrammen en dergelijke.
Door het fixeren van beslissingstabellen op hun (al dan niet vermeende) kracht in het representeren van procedures, werd echter uit het oog verloren dat beslissingstabellen niet in de laatste plaats een techniek voor het doelgericht organiseren c.q. structureren van *gegevens* zijn. Door hun tweeledige oorzaak/gevolg-structuur kunnen beslissingstabellen met name in de initiële, gebruikersgerichte fasen van de levenscyclus van een informatiesysteem vaak met succes worden toegepast.

In dit artikel worden de toepassingsmogelijkheden van beslissingstabellen in diverse fasen van de systeemlevenscyclus besproken en met praktische voorbeelden geïllustreerd. Vooraleer hieraan toe te komen, bakenen we in de tweede paragraaf het concept 'beslissingstabel' wat strikter af dan in tal van publikaties gebruikelijk is. In de derde paragraaf rekenen we af met enige misvattingen omtrent het gebruik van beslissingstabellen als programma-representatietechniek.

2. Het begrip 'beslissingstabel'

Wat onder een beslissingstabel wordt verstaan, is niet altijd even eenduidig vastgelegd. Niet zonder reden definiëren wij een beslissingstabel als 'een tabel waarin een bepaald probleem wordt beschreven via een uitputtende opsomming van elkaar uitsluitende uitspraken' (gecit. uit (21)). Om deze definitie toe te lichten, gaan we uit van de volgende uit (21) overgenomen beslissingstabel:

Orderverwerking					
Type klant	Overheid	Niet-overheid			
Aantal jaren klant (A)	-	$A < 2$	$2 \leqslant A < 5$		$A \geqslant 5$
Afname per jaar < 5000 guldens	-	-	ja	nee	-
% korting	15	0	5	10	15
Factuur in x-voud, x =	5	3	3	3	3
Prioriteitsbehandeling	-	-	-	X	X

Deze beslissingstabel behandelt de situatie 'Orderverwerking' door hierover vijf individuele uitspraken te doen in de vorm van beslissingskolommen. Zo stelt de tweede beslissingskolom dat aan een klant die geen overheidsinstelling is en die minder dan twee jaar klant is, geen korting wordt toegekend en dat een factuur in drievoud dient te worden opgesteld.

Om aan de gegeven definitie van *beslissingstabel* te voldoen, dient een dergelijke conditie/actie (oorzaak/gevolg)-tabel:

a. een *uitputtende* opsomming te bieden. Dit betekent dat alle mogelijke individuele uitspraken in de tabel dienen te worden opgenomen. In concreto houdt deze eis in dat alle relevante condities met hun volledig spectrum aan geldige toestanden en alle mogelijke combinaties van deze geldige toestanden in de beslissingstabel dienen te worden opgenomen. Merk op dat deze eis in vele inleidende en algemene werken, bijv. in (2), niet wordt gesteld. Zoals later zal worden aangetoond, betekent het loslaten van deze eis vooral bij toepassing in de initiële fasen van de systeemlevenscyclus een verschraling van de kracht van de beslissingstabellentechniek.

b. *elkaar uitsluitende* individuele uitspraken te bevatten, wat inhoudt dat geen twee of meer kolommen van een beslissingstabel tegelijk van toepassing kunnen zijn; dit betekent dat alle beslissingskolommen voor tenminste één conditie elkaar uitsluitende toestanden bevatten. Tegelijk eisen we dat de toestanden van elke opgenomen conditie elkaar wederzijds dienen uit te sluiten.

Het gecombineerd optreden van deze eisen van uitputtendheid en exclusiviteit maakt de kracht van een beslissingstabel uit. Door de eis van uitputtendheid verzekeren we ons van het feit dat een probleemsituatie verifieerbaar volledig wordt beschreven, terwijl de exclusiviteitseis garant staat voor een contradictie-vrije, consistente beschrijving van de probleemstelling.

Eén en ander kan ook op een meer *formele* wijze worden gedefinieerd. Beschouw hiertoe NC condities:

$$\{ C_1 , C_2 , C_3 \ldots C_{NC} \}$$

Elke conditie C_i kan m_i disjuncte waarden hebben, namelijk

$$CW_i = \{ cw_{i1} , cw_{i2} \ldots cw_{im_i} \}$$

zodat het testen van C_i neerkomt op het selecteren van één (en slechts één) van de elementen uit CW_i.

Analoog beschouwen we NA acties:

$$\{ A_1 , A_2 , A_3 \ldots A_{NA} \}$$

met voor elke A_i n_i disjuncte waarden, namelijk

$$AW_i = \{ aw_{i1} , aw_{i2} \ldots aw_{in_i} \}$$

In (17) wordt een beslissingstabel gedefinieerd als de *afbeelding* van het Cartesische produkt

$$CW_1 \times CW_2 \times \ldots \times CW_{NC}$$

op het Cartesische produkt

$$AW_1 \times AW_2 \times \ldots \times AW_{NA}$$

De individuele elementen van deze afbeelding noemen we de (enkelvoudige) beslissingskolommen.
Door een beslissingstabel als een mathematische afbeelding te definiëren, leggen we automatisch vast dat elk element van het Cartesische produkt van de conditieruimte één en slechts één beeld in de actieruimte heeft: we werken met 'volledige' beslissingstabellen met wederzijds exclusieve beslissingskolommen.
Een dergelijke formele definitie maakt het mogelijk om allerhande algoritmen uit de besliskunde, de grafentheorie, de algebra van Boole etc. toe te passen op het computer-ondersteund construeren en manipuleren van beslissingstabellen. Voorbeelden van deze algoritmen kunnen worden gevonden in (16), (17) en (20). In paragraaf 8 wordt op dit aspect teruggekomen.

3. De beslissingstabel als algemeen programmaschema

Wanneer we de uitgebreide literatuur over beslissingstabellen[1] doornemen, dan valt op dat een overgroot gedeelte wordt besteed aan het zogenaamde 'omzettingsvraagstuk'. Uitgangspunt van dit vraagstuk is dat (tweedimensionale) beslissingstabellen hoofdzakelijk ter voorbereiding van (n-dimensionale) computerprogramma's worden opgesteld, ofschoon de resultaten van de omzetting van beslissingstabellen in programma's in principe ook

voor het manuele gebruik van beslissingstabellen relevant kunnen zijn. Door een uitgekiende keuze van de volgorde waarin de condities worden getest, proberen de verschillende omzettingsalgoritmen het gemiddelde aantal conditietesten of, meer in het algemeen, de *uitvoeringstijd* van het resulterende programma te minimaliseren.
Nu kan gemakkelijk worden aangetoond dat dit vraagstuk een NP-hard probleem is, wat inhoudt dat de *omzettingstijd* (de tijd die een algoritme nodig heeft om een optimale testvolgorde te bepalen) exponentieel stijgt met het aantal condities. Aanvankelijk werden dan ook vrijwel uitsluitend optimumbenaderende algoritmen gepubliceerd. Toepassing van 'branch and bound'-, dynamische en heuristische programmeringstechnieken leidden tot optimumvindende algoritmen, die voor de omzetting van niet al te uitgebreide beslissingstabellen in mindere of meerdere mate volstaan. De bekendste algoritmen zijn deze van Reinwald en Soland (32), Bayes (1), Schumacher en Sevcik (33), Martelli en Montanari (25), Lew (8) en Papakonstantinou (30). Onderling verschillen deze algoritmen niet alleen in de toegepaste technieken, maar vooral ook in de randvoorwaarden die ze bij de omzetting in acht nemen. Hierdoor is het ook bijzonder moeilijk om deze algoritmen onderling te gaan vergelijken.
In geen enkele publikatie werd aanvankelijk ook maar een kanttekening geplaatst bij het relatieve belang van deze omzettingsalgoritmen. Chvalovsky (3) was ons inziens de eerste die heeft gewezen op de overbeklemtoning van het omzettingsvraagstuk ten koste van artikelen over ervaringen met beslissingstabellen in praktijksituaties.

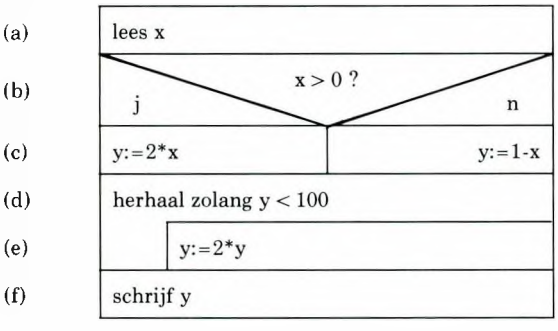
De beperkte relevantie van het omzettingsvraagstuk wordt door de volgende omstandigheden in de hand gewerkt:

1. Condities zijn veelal niet onafhankelijk van elkaar en bijgevolg kunnen veel van de vermelde algoritmen niet integraal worden toegepast. Immers, sommige condities kunnen niet worden getest indien andere ('hogere') condities hetzij nog niet werden getest, hetzij een bepaalde waarde aannemen. Zo kan de vraag of de dagdatum al dan niet geldig is enkel worden beantwoord als we vooraf al de maanddatum hebben bepaald en als deze een geldige (tussen 1 en 12) waarde heeft.
   Dit alles betekent dat (in dit geval) de conditietest over de dagdatum nooit vóór de conditietest over de maanddatum kan worden uitgevoerd. Hierdoor wordt de permutatie van mogelijke testvolgorden aanzienlijk ingeperkt en bijgevolg ook de gemiddelde winst die door het gebruik van optimalisatie-algoritmen kan worden behaald.
2. Voor gebruiksintensieve programma's komen enkel optimumvindende algoritmen in aanmerking. Deze algoritmen vereisen echter alle informatie over de testtijden van de verschillende condities en over de frequenties waarmee de beslissingskolommen kunnen optreden. Vrijwel altijd en zeker bij het initiële ontwerp van de beslissingstabel ontbreekt hierover de nodige informatie. Dit betekent dat deze zogenaamde optimumvindende algoritmen zelfs in afwezigheid van onderling gerelateerde condities praktisch beschouwd slechts een pseudo-optimalisatie kunnen bewerkstelligen.

3. Door de steeds hogere verwerkingssnelheid van moderne computers daalt de relevantie van het vinden van kortere uitvoeringstijden. Bovendien moet de relatieve winst worden afgewogen tegen de omzettingstijd van het algoritme zelf; deze laatste stijgt, zoals gesteld, exponentieel of meer met het aantal condities.
4. De gemiddelde verwerkingssnelheid van het resulterende programma is slechts één mogelijk, zij het een belangrijk evaluatiecriterium. Andere criteria betreffen onder meer de minimalisatie van de 'worst case'-uitvoeringssnelheid of van de gebruikte geheugenruimte. Moret e.a. (28) hebben aangetoond dat in het algemeen niet gelijktijdig aan meerdere van deze criteria kan worden voldaan.

Een veel belangrijker argument wordt echter ontleend aan het feit dat de inherente premisse van deze algoritmen, namelijk dat beslissingstabellen algemeen geschikt zijn voor het representeren van complexe programmastructuren, in de praktijk onjuist blijkt te zijn.

Beschouw bijvoorbeeld volgend (overigens triviaal) Nassi-Shneiderman-diagram:



Het is duidelijk dat de zes instructieregels logisch bij elkaar horen; er treedt geen enkele structuurbreuk op, wat aanleiding zou kunnen geven tot het onderbrengen van een deel van het programma in een aparte module. Toch vereist de omzetting van dit eenvoudige programma een drievoudige beslissingstabellenstructuur: één tabel met instructieregel (a), één tabel met de instructieregels (b) en (c) en één tabel met de instructieregels (d), (e) en (f). Door de beperkingen van de beslissingstabel (in wezen niet meer dan een complexe selectiestructuur), moeten we bijgevolg een kunstmatige programmastructuur aanbrengen. Dit houdt in dat we voor elke actie/conditie-opeenvolging (hier tussen (a) en (b) respectievelijk tussen (c) en (d)) een nieuwe beslissingstabel dienen te gebruiken.

In de literatuur werd het probleem impliciet onderkend en tal van alternatieve conventies voor het representeren van beslissingstabellen werden voorgesteld. Deze alternatieven werden uitvoerig besproken in (15); alle, inclusief het recente voorstel van Mc Mullen (27), leveren ze meer nadelen dan voordelen op. Eén alternatief kan hier niet onvermeld blijven, al was

het maar om de nadruk waarmee het door zijn ontwerper wordt gepropageerd (zie bijv. (9), (10), (11) en (12)). Zo wordt in (9) aangetoond dat elk programma in wezen tot één beslissingstabel kan worden gereduceerd, mits we een globale variabele δ (met als standaard startwaarde 0) invoeren en de conventie aanhouden dat elke beslissingskolom, indien niet expliciet van een EXIT-instructie voorzien, 're-entrant' werkt. Op deze wijze kan het zojuist gegeven voorbeeld door middel van de volgende beslissingstabel worden weergegeven:

δ =	0	1		2	
x > 0	-	j	n		
y < 100	-	-	-	j	n
lees x	x	-	-	-	-
y :=	-	2*x	1-x	2*y	-
schrijf y	-	-	-	-	x
δ :=	1	2	2	-	-
EXIT	-	-	-	-	x

Dit voorstel komt er in feite op neer dat de drie benodigde beslissingstabellen naast elkaar worden geschreven, met één globale variabele (δ) als regelende instantie. Duidelijk is dat de in de vorige paragraaf vermelde voordelen van het gebruik van beslissingstabellen (controle op volledigheid en dergelijke) hier quasi volledig verloren gaan.

Met deze vrij negatieve, voorlopige conclusie als leidraad, zullen we in de volgende paragraaf nagaan in hoeverre beslissingstabellen als programmeertechniek alsnog enige waarde behouden. In paragraaf 5 bespreken we afzonderlijk het gebruik van beslissingstabellen voor het genereren van testdata.

4. De beslissingstabel als programmeertechniek

Als we rekening houden met de conclusies van de voorgaande paragraaf, kunnen we het gebruik van beslissingstabellen als *programmeertechniek* zoals we dat in de literatuur beschreven vinden, tot de volgende drie punten reduceren:

1. Vooreerst kan een aantal problemen waarvoor programma's dienen te worden ontworpen, wel degelijk in de vorm van een complexe selectiestructuur worden beschreven. Typische voorbeelden betreffen het lexicale analysegedeelte van een compiler, programma's voor inputvalidatie en dergelijke. In (17, pag. 104 e.v.) is een lijst van representatieve problemen opgenomen en worden de voordelen verbonden aan het structureren in beslissingstabellenvorm opgesomd. Voor deze beperkte lijst van problemen is het gebruik van beslissingstabellen als algemeen programma-organisatieschema bijgevolg aangewezen.

2. De beslissingstabel kan uitstekend als een *deelstructuur* van een programma worden beschouwd. In deze gesteldheid blijven twee toepassingsmogelijkheden open:

   A. De beslissingstabel wordt, bijvoorbeeld als een *decision*-statement, aan de instructieset van een programmeertaal toegevoegd. Een overigens niet zo elegant voorstel hiervoor werd ontwikkeld in (17). In (35) stelt Vanthienen een alternatief *dectable*-statement voor. Deze instructie maakt het mogelijk om de specificatie van een beslissingstabel in de vorm van een *inverted select* in een regulier (PASCAL-)programma op te nemen. Door middel van een precompiler, die overigens op een verfijning van het PRODEMO-systeem (zie paragraaf 8) is gebaseerd, wordt deze specificatie automatisch in de overeenkomstige beslissingstabel omgezet en in deze vorm als commentaar aan de programmatekst toegevoegd. Hoe één en ander er uitziet, kan uit onderstaand eenvoudig voorbeeld worden afgeleid (overgenomen uit (35), pag. 73):

```
DECTABLE
    COND    1 : Afstand : < 5, > =5;
            2 : Bedrag : < 10, > =10;
            3 : Belangrijke Klant : Y, N;
    ACT     1 : WRITELN ('Vervoer');
            2 : WRITELN ('Toeslag');
            3 : WRITELN ('Geen kosten');
    RULES   1 : 1 indien (1a en 2a en 3b) of 1b;
            2 : 2 indien 1a en 2a;
            3 : 3 indien niet (regel 1 of regel 2)
END

(*
```

1. Afstand	< 5			> =5
2. Bedrag	< 10		> =10	–
3. BelangrijkeKlant	Y	N	–	–
1. WRITELN ('Vervoer')		x		x
2. WRITELN ('Toeslag')	x	x		
3. WRITELN ('Geen kosten')			x	

```
*)
```

   B. De beslissingstabel wordt, met het oog op programma-documentatie, als een compacte en bijgevolg overzichtelijke selectiestructuur in een Nassi-Shneiderman-diagram ingelast. Het volgende voorbeeld, licht gewijzigd overgenomen uit (36), bewijst dat een beslissingstabel in sommige gevallen is te verkiezen boven een geneste IF-THEN-ELSE-structuur:

Herhaal voor elke order				
Kredietlimiet overschreden	J	N	N	N
Voorraad voldoende	–	J	N	N
Voorraad = 0	–	–	J	N
Order hangend houden	X	–	–	–
Order volledig leveren	–	X	–	–
Op wachtlijst plaatsen	–	–	X	–
Gedeeltelijk leveren, gedeeltelijk op wachtlijst	–	–	–	X
Bestelgegevens bijwerken	–	X	X	X
Leveringsgegevens bijwerken	–	X	–	X

3. Het omzetten van programma's in beslissingstabellenvorm kan als een maatstaf voor het *meten van programmacomplexiteit* worden gehanteerd ((9), (17) en (23)). De uitgangspunten van dit op het eerste gezicht wat vreemd aandoend gebruik zijn:
   - de overweging dat de logische complexiteit (dit is de complexiteit van de besturingsstructuur) een goede maatstaf voor de totale complexiteit van een programma is;
   - de overtuiging dat de beslissingstabel een goed ordeningsschema voor de menselijke logica is.

Bijgevolg wordt in de gerefereerde publikaties gesteld dat het gemak waarmee programma's (al dan niet fictief) in een beslissingstabellenvorm kunnen worden gegoten een maat is voor de logische complexiteit en ergo voor de totale complexiteit van een programma. In (17) en (23) wordt dit gemak vertaald in een koppel (N(T) : V(T)), waarbij:
- de ondergrens N(T) wordt gevormd door het aantal beslissingstabellen dat nodig is om een bepaald programma(gedeelte) te representeren;
- de bovengrens V(T) wordt, in navolging van het cyclomatisch getal van Mc Cabe (26), gelijk gesteld aan het totaal aantal (over de verschillende beslissingstabellen verspreide) kolommen minus het aantal tabellen plus 1. In formulevorm:

$$V(T) = N(R) - N(T) + 1$$

In (23) wordt deze maatstaf vergeleken met een groot aantal gepubliceerde alternatieven. Opvallend is dat de maatstaf bijzonder goed blijkt overeen te stemmen met de intuïtieve notie van programmacomplexiteit, zoals deze door een programmeur wordt ervaren. Uiteraard dienen deze voorlopige resultaten nog verder te worden getoetst aan praktijkvoorbeelden.

5. De beslissingstabel en het testen van programma's

In weerwil van alle programmeerhulpmiddelen blijft de betrouwbaarheid van de software nog steeds één van de Achillespezen van het automatiseringsgebeuren. Een beproefde methode voor het verhogen van deze be-

trouwbaarheid is het onderwerpen van de software aan een aantal testgevallen[3]. Via deze testgevallen kunnen we nooit de formele correctheid van een programma aantonen; wel kunnen we door het uitvoeren van een gestratificeerde verzameling testen het resterend aantal fouten proberen te minimaliseren. Deze gestratificeerde verzameling omvat testen die op verschillende niveaus (bijv. per module, per programma en per systeem), door verschillende personen (bijv. eindgebruikers en programmeurs) en op verschillende tijdstippen (bijv. bij het module-ontwerp, bij de integratie van programma's tot informatiesystemen etc.) dienen te worden uitgevoerd.

Het gebruik van beslissingstabellen bij het genereren van testgevallen lijkt voor de hand te liggen: per beslissingskolom genereren we één testgeval. Zowel de gebruikers (op basis van gebruikers-systeemspecificaties), de ontwerpers (op basis van ontwerpspecificaties) als de programmeurs (op basis van programmaspecificaties en programmalogica) kunnen deze werkwijze zonder meer toepassen. Nog afgezien van het feit dat formele specificaties slechts heel zelden voorhanden zijn, wensen wij in wat volgt enige kanttekeningen te plaatsen bij dit vaak verkondigd, maar ons inziens te weinig genuanceerd standpunt. Hierbij gaan we uit van de tweedeling in 'white box'- en 'black box'-strategieën voor het genereren van testgegevens. Eerstgenoemde maken gebruik van de interne programmastructuur om de kans op foutendetectie te maximaliseren. 'Black box'-strategieën daarentegen gebruiken hiertoe externe programma-specificaties.

5.1 'White box'-strategieën

Voor het genereren van testdata volgens het 'white box'-principe geeft Myers ((29), pag. 37 e.v.) een reeks criteria met oplopende dekkingsgraad. Het strengste criterium betreft de 'multiple condition'-dekking, waarbij, samengevat, voldoende testgevallen worden gegenereerd om alle mogelijke combinaties van conditie-uitkomsten in elk beslissingspunt ten minste één keer te doorlopen. Zo geeft het beslissingspunt

while (j<0) *and* (k=1)

aanleiding tot het genereren van de volgende verzameling testgevallen:

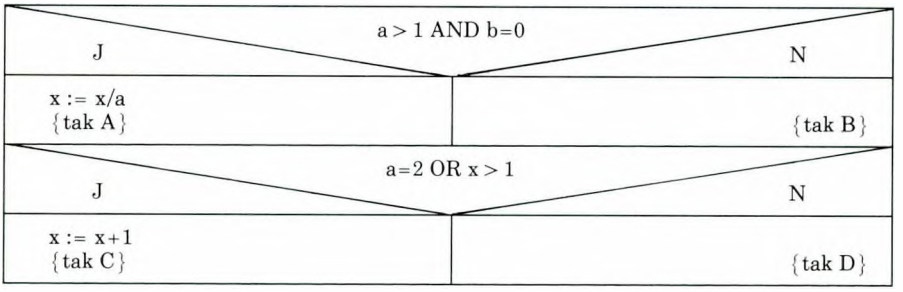
1) (j = -1) en (k = 1)
2) (j = 0) en (k = 2)
3) (j = -1) en (k = 2)
4) (j = 0) en (k = 1)

Zetten we dit voorbeeld om in beslissingstabellenvorm, dan geeft dit:

j<0	J		N	
k=1	J	N	J	N

wat aanleiding geeft tot het ontwerpen van dezelfde 4 testgevallen (één per kolom): de beslissingstabel voldoet aan de verwachtingen!

In het algemeen kan echter worden gesteld dat het toepassen van het 'multiple condition'-criterium (bijvoorbeeld via de omzetting in beslissingstabellen) in het geval van logisch complexe modules met relatief veel samengestelde beslissingspunten vrij snel kan leiden tot een combinatorisch exploderende en bijgevolg onuitvoerbare verzameling testgevallen.
Myers opteert ook in het geval van meerdere beslissingspunten voor 'multiple condition coverage'. Hij geeft volgend voorbeeld ((29), pag. 39; hier echter omgezet in een Nassi-Shneiderman-diagram):



met als testgevallen (inclusief de doorlopen takken):

1) (a = 2), (b = 0) en (x = 4)	{ A - C }
2) (a = 1), (b = 1) en (x = 1)	{ B - D }
3) (a = 2), (b = 1) en (x = 1)	{ B - C }
4) (a = 1), (b = 0) en (x = 2)	{ B - C }

Uit de doorlopen takken kan worden opgemaakt dat het 'multiple condition' dekkingscriterium niet noodzakelijk leidt tot het testen van alle mogelijke combinaties van moduletakken ({A - D} ontbreekt, terwijl {B - C} twee keer wordt doorlopen).
Om hieraan te verhelpen stellen wij de volgende alternatieve, op het gebruik van beslissingstabellen gebaseerde methode voor het genereren van 'white box'-testgevallen voor:

1. Genereer testgevallen voor alle mogelijke combinaties van moduletakken[4]. Hiervoor is het voldoende een beslissingstabel op te stellen met één conditie per beslissing. Deze tabel is de testtabel op moduleniveau.
   Toegepast op bovenstaand voorbeeld geeft dit[5]:

α	J		N	
β	J	N	J	N
{takken}	A - C	A - D	B - C	B - D

2. Voor elke samengestelde conditie (hier voor beide condities) construeren we een hiërarchisch ondergeschikte beslissingstabel. Dit geeft bijvoorbeeld voor α:

a > 1	J		N	
b=0	J	N	J	N
α :=	J	N	N	N

3. We kiezen vervolgens de testgevallen zodanig dat in elk geval aan het 'multiple decision coverage'-criterium is voldaan. Dit houdt in dat we de testtabel op moduleniveau kolomsgewijs doorlopen. Voor de hiërarchisch ondergeschikte tabellen kunnen we, afhankelijk van ons ambitieniveau, opteren voor 'multiple condition coverage' (en bijgevolg voor elke kolom een testgeval genereren) of voor het genereren van één testgeval per uitkomst (hier: één voor α=J en één voor α=N).

Als slot nog deze drie opmerkingen:
- Een belangrijk probleem waar ook beslissingstabellen zonder nut zijn, blijft het terugrekenen van de conditietoestanden naar de geschikte waarden voor de inputvariabelen. Op meer lokale schaal dienen we ons overigens bewust te zijn van het feit dat de variabelen tussen de verschillende beslissingspunten van waarde kunnen veranderen (in bovenstaand voorbeeld: de variabele a zou tussen de α- en β-beslissingspunten een nieuwe waarde kunnen krijgen).
- De hier geschetste methode is fundamenteel verschillend van de benadering van Walsh (37), die ook uitgaat van het gebruik van beslissingstabellen. Naar onze mening dient het gestructureerd testen van modules top down te geschieden. Deze top down benadering wordt weerspiegeld in de hiërarchie van beslissingstabellen. Walsh daarentegen vertrekt van het concept 'unit testing', waarbij een 'unit' in haar termen een functioneel bijeenhorend, zeer klein gedeelte van een module is. Voor het testen van de volledige module voegt Walsh een aantal van deze geteste 'units' samen; bij deze tweede stap van haar bottom up methode maakt Walsh verder geen gebruik van beslissingstabellen, waardoor zij geen 'multiple decision coverage' kan garanderen.
- In de sub 4.1 vermelde (in feite: 'one decision/multiple condition'-) gevallen kunnen beslissingstabellen zonder problemen worden gebruikt voor het genereren van testgegevens. Toch moet ook hier worden gewaarschuwd voor een te uitgebreide verzameling testgevallen.

5.2 'Black box'-strategieën

Bij het ontwerpen van 'black box'-strategieën wordt uitgegaan van equivalentieklassen. Dit zijn homogene verzamelingen van waarden van inputvariabelen die door het programma op identieke wijze zullen worden verwerkt; ze worden opgesteld aan de hand van (externe) programmaspecificaties. Door voor elke equivalentieklasse één representatieve waarde te kiezen, kunnen we het aantal testgevallen relatief beperken.

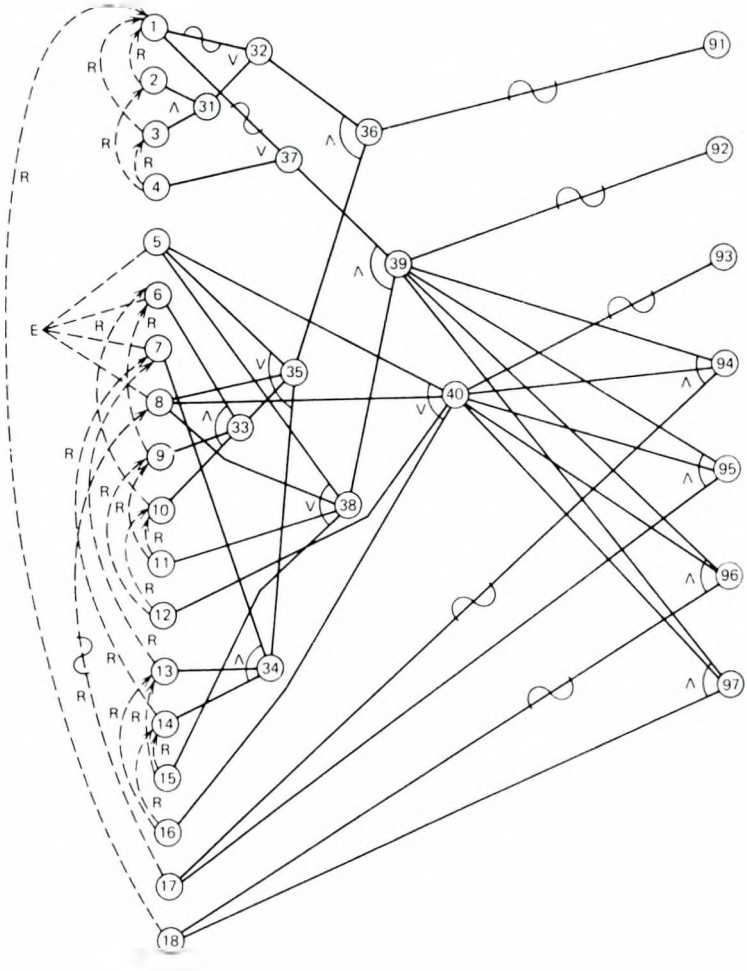
Voorbeeld
Voor het invullen van de belastingformulieren wordt rekening gehouden met een aantal personen ten laste variërend van 0 tot 12. De drie bijbehorende equivalentieklassen zijn:

$$< 0 / 0 - 12 / > 12$$

Als representatieve waarden kunnen wij bijvoorbeeld nemen:

$$-1 / 12 / 13$$

Zonder hier in detail in te gaan op het uit de equivalentieklassen afleiden van het minimaal aantal testgevallen, kunnen we niettemin stellen dat een belangrijk probleem is gelegen in het aanbrengen van de *relaties* tussen de inputvariabelen (en tussen de outputvariabelen). Myers ((29), pag. 56-73) gebruikt hiertoe 'cause-effect graphs'. Een voorbeeld van een (overigens nog relatief eenvoudige) graph ontlenen we aan (29), pagina 66:



In een verdere stap wordt deze graph omgezet in een (bij Myers: limited-entry[6]) beslissingstabel, waarna in een laatste stap testgevallen (één per kolom) worden gegenereerd.
Ofschoon een betere kennis van de verschillende gebruikte symbolen ongetwijfeld verhelderend werkt, toch kan niet worden ontkend dat het opstellen en interpreteren van een dergelijke spinnewebachtige graph geen sinecure is. In (18) stellen Maes en Mercken dan ook voor om het ontwerpen van de graph te vervangen door het rechtstreeks construeren van de (extended-entry) beslissingstabel. Hiervoor kunnen gestandaardiseerde constructiemethoden worden gevolgd en kan eventueel een beroep worden gedaan op computerondersteuning. Een en ander wordt in paragraaf 8 nader toegelicht.

Tot slot dient te worden opgemerkt dat ook Goodenough en Gerhart (6) beslissingstabellen gebruiken voor het representeren van testgevallen. In tegenstelling tot wat hier wordt voorgehouden, zijn beslissingstabellen bij hen niet zozeer een ontwerp-, maar veeleer een loutere representatietechniek.

6. De beslissingstabel en de initiële systeemontwikkelingsfasen

Moderne systeemontwikkelingsmethodieken benadrukken het belang van de initiële ontwikkelingsfasen[7]: door het zorgvuldig doorlopen van deze gebruikersgerichte fasen worden niet alleen effectievere informatiesystemen gebouwd, maar bovendien wordt hierdoor ook de psychologische betrokkenheid van de eindgebruiker beduidend verhoogd.

Beslissingstabellen, zo blijkt uit de praktijk, kunnen in deze fasen in verschillende hoedanigheden worden ingezet. In wat volgt beperken we ons tot een vrij summiere bespreking. Een meer uitgebreide beschrijving met tal van uitgewerkte voorbeelden kan worden gevonden in (5), (17) en (36).
Cheng en Rabin (4) hebben er als eersten op gewezen dat beslissingstabellen als '*fact finding*'-techniek kunnen worden ingezet; hierbij wordt in de eerste plaats een beroep gedaan op het feit dat beslissingstabellen een gegeven situatie uitputtend beschrijven. In deze hoedanigheid zijn ze nuttig zowel bij de activiteitenanalyse als bij de beslissingsanalyse.
Gebleken is dat beslissingstabellen een uitstekend instrument vormen voor het gestructureerd analyseren van logisch complexe situaties. Zo wordt in (22) verwezen naar een uitgebreide analyse van het beslissingsproces dat zich afspeelt bij het al dan niet toekennen van een rekening-courant: beslissingstabellen vormen in dergelijke situaties een uitstekend *communicatiemedium* tussen de eindgebruiker en de informatie-analyst. In (17) wordt aangetoond dat een gestructureerde beschrijving van een activiteiten- en/of beslissingssituatie in beslissingstabellenvorm bovendien een potentieel uitgangspunt vormt voor het deduceren van de informatiebehoeften.
Ook Grindley (7) gebruikt beslissingstabellen voor het analyseren en repre-

senteren van beslissingen. Hierbij beperkt hij zich echter tot individuele beslissingspunten met slechts één actie per tabel. Typerend is verder dat hij de condities opsplitst in gewone (in termen van Grindley: 'subsidiary') condities en zogenaamde 'triggering' condities die specificeren wanneer de uitvoering van de beslissingstabel dient te worden geïnitieerd.
Beslissingstabellen kunnen verder, zoals onder meer door Lundeberg c.s. (13) in de fase informatie-analyse van de ISAC-methodiek wordt aangetoond, worden gebruikt voor de functionele, logische *specificatie van de informatie (verwerkende) processen*. Deze processen kunnen al dan niet uit de geanalyseerde activiteiten worden afgeleid. In deze hoedanigheid wordt vooral gebruik gemaakt van het feit dat de kolommen van de beslissingstabel elkaar wederzijds uitsluiten, waardoor een eenduidige beschrijving van de informatieprocessen wordt gewaarborgd. Complexe voorbeelden in deze sfeer kunnen worden gevonden in (17) en vooral (36).

Samenvattend kan worden gesteld dat het gebruik van beslissingstabellen in de initiële fasen van de systeemontwikkeling in feite alleen wordt beperkt door opportuniteitsoverwegingen. Beslissingstabellen die gedurende de activiteiten- en beslissingsanalyse worden opgesteld, kunnen veelal eenvoudig worden omgezet in tabellen die de informatieprocessen beschrijven. Deze tabellen vormen op hun beurt een goed uitgangspunt voor het aanleveren van de programmaspecificaties. Een en ander wordt door Verhelst (36) uitvoerig geïllustreerd.

7. De beslissingstabel en het nemen van proceduurmatige beslissingen

Hetgeen in voorgaande paragraaf werd gesteld met betrekking tot het gebruik van beslissingstabellen in de initiële fasen van de systeemontwikkeling, geldt ook vrijwel integraal voor proceduurmatige beslissingen in het algemeen. Beslissingstabellen betekenen voor dit type beslissingen vaak een uitstekend communicatiemedium tussen de opsteller van de te volgen procedures en de uitvoerder van de procedures. Typische voorbeelden betreffen wetteksten, diverse handleidingen en voorschriften, etc. Dit houdt in dat beslissingstabellen ook in de *operationele fase* van een informatiesysteem kunnen worden gebruikt.

Beslissingstabellen zijn, in tegenstelling tot vrijwel alle voorschriften in tekstvorm, conditie-georiënteerd. Dit betekent dat eerst alle relevante condities worden getest vooraleer het geheel van de toe te passen acties wordt uitgevoerd. Hierdoor kan in een concrete beslissingssituatie niet alleen het testen van irrelevante condities worden vermeden, maar bovendien dient elke conditie maar één keer te worden geëvalueerd: een beslissingstabel wordt van boven naar onder gelezen. Dit houdt in dat het nemen van beslissingen volgens een door middel van een beslissingstabel beschreven procedure *sneller* kan geschieden en dat er *minder* mogelijkheden tot het insluipen van *fouten* worden geschapen.

8. Het computerondersteund construeren en manipuleren van beslissingstabellen

Beslissingstabellen, zoveel is duidelijk, worden steeds meer als hulpmiddel bij de analyse en minder bij het programmeren gebruikt. Een opvallend kenmerk van de analysefase is haar iteratief karakter: in een aantal opeenvolgende beschrijvingen wordt geprobeerd de werkelijkheid te vatten. Dit betekent dat de gebruikte representatiemiddelen, onafhankelijk van het feit of het nu beslissingstabellen, gegevensstroomdiagrammen, ISAC-schema's of wat ook zijn, veelvuldig dienen te worden overgetekend. Het wekt dan ook geen verwondering dat alom naar computerondersteuning voor het constructieproces wordt uitgekeken.
Verhelst was de eerste die het belang van gestandaardiseerde methoden voor het construeren van beslissingstabellen als basis voor computerondersteuning heeft ingezien. In (36) werkt hij eerder door hem gesuggereerde methoden uit tot de zogenaamde 'directe methode' en de 'zoekmethode'. In het CODASYL-rapport over beslissingstabellen (5) worden zijn voorstellen vrijwel integraal overgenomen.
Het uitgangspunt van de genoemde methoden is dat het constructieproces in een drietal fasen dient te worden opgesplitst:

1. Het opstellen van een lijst van relevante condities (met hun successieve toestanden) en acties.
2. Het formeel beschrijven van de logische wetmatigheden die de probleemsituatie beheersen.
3. Het feitelijk opstellen van de beslissingstabel (eventueel gevolgd door het samentrekken ervan).

De *directe constructiemethode* gaat uit van een gedetailleerde probleembeschrijving op papier: het opstellen van de lijst van relevante condities en acties kost in dit geval niet al te veel moeite. Voor het beschrijven van de wetmatigheden opteert Verhelst voor een intermediaire vorm met AND-, OR- en NOT-operatoren. Een voorbeeld ((36), pag. 56):

$$[(KI \leqslant 10)\ EN\ (B < 5)]\ OF\ \{[(KI \leqslant 10)\ OF\ (10 < KI \leqslant 12)]\ EN\ (5 \leqslant B < 30)\}\ OF\ [NOT(KI > 20)\ EN\ (B \geqslant 30)] \rightarrow A2$$

Hoewel de cryptische formulering van deze voorwaardelijke uitspraken op het eerste gezicht eerder in de omgekeerde richting wijst, kan in complexe situaties niettemin worden vastgesteld dat deze uitspraken een goed houvast bieden bij het formaliseren van de probleemlogica. Het opstellen van de tabel (fase 3) gebeurt vervolgens door deze geformaliseerde uitspraken rechtstreeks in het actiegedeelte van de geëxpandeerde tabel in te vullen.
De *zoekmethode* wordt gebruikt indien geen geschreven probleemformulering voorhanden is. In dit geval worden de drie fasen van het constructieproces iteratief doorlopen. Vanzelfsprekend verloopt dit proces, waarbij we zowel met betrekking tot de op te nemen condities en acties als de probleemlogica gedeeltelijk in het duister tasten, minder strak en geformaliseerd dan in het voorgaande geval: het betreft in hoge mate een 'trial and error'-proces.
Beide constructiemethoden worden integraal ondersteund door het PRO-

DEMO-systeem[8]. Dit gebruiksvriendelijk en flexibel softwarepakket voor het construeren en manipuleren van beslissingstabellen heeft tot doel de gebruiker ervan te begeleiden en te steunen door het controleren van de juistheid en volledigheid van de beslissingstabel, door het uitvoeren van tijdsintensieve manipulaties zoals het maximaal samentrekken van de tabel, door het tekenen van de tabel en door het geven van aanbevelingen met betrekking tot de te volgen stappen gedurende het modelleerproces. In (19) en (22) wordt uitvoerig ingegaan op zowel de interne structuur als het gebruik van PRODEMO. In wat volgt beperken we ons tot een bespreking van de hoofdlijnen.

Het systeem is gebaseerd op een krachtig intern representatieschema (de zogenaamde 'uitgebreide decision grid chart', zie (17)) en op algoritmen voor het manipuleren van zowel dit interne schema als van de beslissingstabel zelf. Een aantal van dergelijke algoritmen wordt in de literatuur beschreven (zie hiervoor bijvoorbeeld (16), (17), (20) en (35)).
Een heel wezenlijk onderdeel van een dergelijk systeem betreft verder een uitgebreide *specificatietaal*, die de eerder primitieve vorm van de voorwaardelijke uitspraken uit de directe methode kan ombuigen in de richting van een werkelijk probleemgerichte specificatie. In (17) wordt aangetoond dat deze taal uit de volgende elementen dient te bestaan:

1. De logische AND-, OR-, NOT- en (eventueel) XOR-operatoren voor het groeperen van zowel de condities als de acties tot meer complexe operanden.
2. Causale operatoren voor het representeren van de relaties tussen de condities en de acties.
3. Limitatoren voor het representeren van de interrelaties tussen de individuele voorwaardelijke uitspraken. Hierdoor wordt het bijvoorbeeld mogelijk 'algemene regel/uitzondering'-situaties op een natuurlijke wijze te beschrijven. De algemene regel wordt in dit geval in één voorwaardelijke uitspraak beschreven, terwijl de voorwaardelijke uitspraken voor de uitzonderingen via een limitator aan deze eerste voorwaardelijke uitspraak worden gerelateerd.
4. Operatoren voor het beschrijven van de onderlinge afhankelijkheden van zowel de condities als de acties. Een voorbeeld van dergelijke afhankelijkheden (in dit geval van condities) werd in paragraaf 3 gegeven (dagdatum versus maanddatum).

De kracht van dit voorstel is gelegen in de zeer eenvoudige structuur van de causale operatoren, die de condities en de acties met elkaar verbinden en in wezen de kern van de voorwaardelijke logica uitmaken. Deze causale operatoren kunnen worden herleid tot de volgende vier basisvormen:

- De *dwingende gevolgtrekking*, dat wil zeggen het gestelde is onbeperkt geldig; elke poging tot het inperken van deze geldigheid resulteert in een foutmelding.
- De *logische gevolgtrekking*, dat wil zeggen het gestelde geldt voor zover het niet wordt tegengesproken of ingeperkt door een andere voorwaardelijke uitspraak.

- De *potentiële gevolgtrekking*, die enkel een mogelijke relatie ('indien . . ., dan kan . . .') tussen de actie- en conditieconfiguratie aangeeft. Om in enige betekenisvolle uitspraak te figureren, dient deze operator te worden gecombineerd met de hierna beschreven intensieve gevolgtrekking.
- De *intensieve gevolgtrekking*, die in tegenstelling tot voorgaande operatoren in het conditie- of actiegedeelte zelf komt te staan. In het eerste geval betekent deze operator 'enkel indien . . .', in het tweede geval 'enkel de actie . . .'.

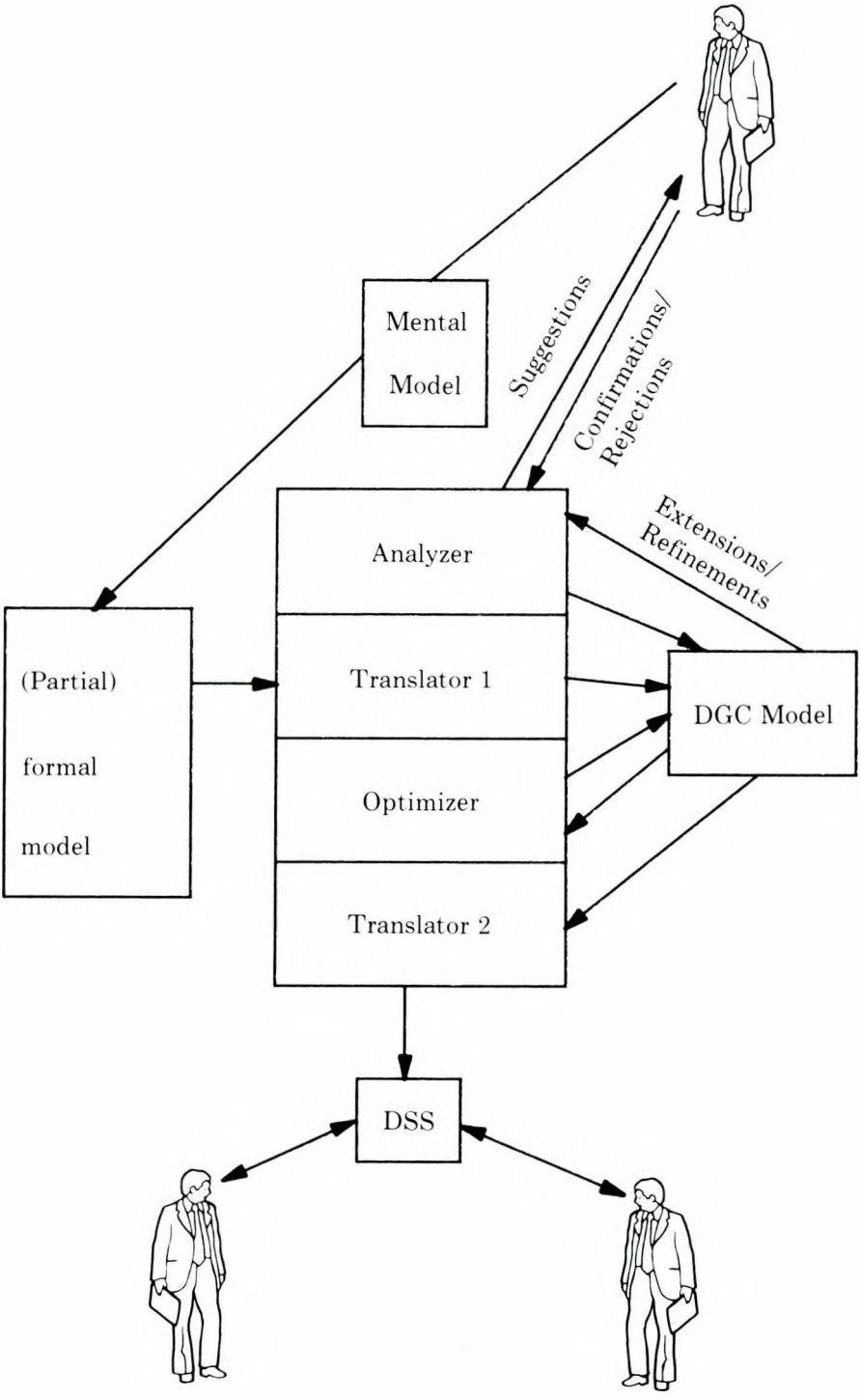
In (17) wordt door middel van voorbeelden aangetoond dat een combinatie van deze operatoren in staat is om elke voorwaardelijke uitspraak op een gebruiksvriendelijke wijze in een formeel toetsbare vorm om te zetten.
Een variant van deze basisideeën is ondertussen geïmplementeerd in het PROLOGA-systeem (zie (35)). In (24) tonen Maes en Van Dijk onder meer aan dat deze specificatietaal en het geassocieerde interne representatieschema kunnen fungeren als de logische component van een expert systeem voor het genereren van beslissingsondersteunende systemen. Op dit aspect wordt in de volgende paragraaf dieper ingegaan.

9. Conclusies en toekomstverwachtingen

De algemene teneur van onderhavig artikel is dat beslissingstabellen in de programmeerfase slechts heel beperkt inzetbaar zijn, maar dat heel wat meer mogelijkheden voor deze techniek open liggen in de initiële, gebruikersgerichte ontwikkelingsfasen. Wij zijn wat langer blijven stilstaan bij het gebruik van beslissingstabellen voor het genereren van testgegevens, omdat dit onderwerp in de literatuur eerder stiefmoederlijk wordt behandeld.
Kijken we wat meer in detail naar de richting waarin het onderzoek in de beslissingstabellentechniek dient te worden georiënteerd, dan kunnen ons inziens de volgende twee belangstellingsgebieden worden afgebakend:

1. De potentiële toepasbaarheid van de techniek in de initiële systeemontwikkelingsfasen, zo hebben we gesteld, wordt op dit ogenblik alleen begrensd door opportuniteitsoverwegingen. Dit betekent dat in grote mate een beroep wordt gedaan op de ervaring en het inzicht van (bijvoorbeeld) de informatie-analyst om te bepalen of hij/zij de techniek in een bepaalde situatie al dan niet kan gebruiken.
   Naar onze mening bestaat er een dringende behoefte aan een wetenschappelijk ondersteunde typologie van toepassingsmogelijkheden. Een eerste aanzet tot een dergelijke typologie zou kunnen bestaan uit het inventariseren van positieve én negatieve ervaringen met het gebruik van beslissingstabellen in real-life situaties.
2. Het gebruik van beslissingstabellen bij het analyseren en beschrijven van proceduurematige beslissingen overstijgt, zo werd in de voorgaande paragraaf gesteld, de directe systeemontwikkelingssfeer. Een vrij logische stap zou bijgevolg kunnen bestaan uit het integreren van het PRODEMO-systeem in de veelbesproken, maar tot op heden relatief weinig toegepaste 'manager's workbench'.

x


Mental
Model
Suggestions
Confirmations/
Rejections
Extensions/
Refinements
Analyzer
(Partial)
formal
model
Translator 1
Optimizer
Translator 2
DGC Model
DSS

In (24) wordt aangegeven hoe althans het basisprincipe van dit systeem kan worden uitgebouwd tot een algemeen toepasbare generator voor beslissingsondersteunende systemen (zie de bijgevoegde figuur, overgenomen uit (24)). Via een uitbreiding van de sub 8. besproken specificatietaal zet de 'decision modeler' zijn mentaal model om in een formeel model, dat vervolgens door toepassing van optimalisatie-algoritmen wordt omgezet in een consistent intern model (de zogenaamde *D*ecision *G*rid *C*hart) van de (partiële) beslissingssituatie. Door toepassing van algoritmen uit de artificiële intelligentie, meer bepaald uit de patroonherkenning, kunnen uit dit partieel model vervolgens suggesties voor aanvulling, verfijning etc. van de voorlopige beschrijving aan de 'decision modeler' worden voorgelegd. Op deze wijze functioneert de generator als een gebruiksvriendelijk expert systeem voor het opbouwen van een consistent en volledig beslissingsmodel, dat vervolgens door de 'decision makers' kan worden gebruikt.

Referenties

1. Bayes, A. J., A dynamic programming algorithm to optimise decision table code, *The Australian Computer Journal*, 5(2), mei 1973, pag. 77-79.
2. Bemelmans, T. M. A., *Bestuurlijke informatiesystemen en automatisering*, Stenfert Kroese, Leiden/Antwerpen, 1984, 309 pag.
3. Chvalovsky, V., Problems with decision tables, *Communications of the ACM*, 19(12), december 1976, pag. 705-707.
4. Cheng, C. W. & Rabin, J., Synthesis of decision rules, *Communications of the ACM*, 18(7), juli 1975, pag. 404-406.
5. CODASYL, *A modern appraisal of decision tables*, Report of the Decision Table Task Group, ACM, New York, 1982.
6. Goodenough, J. B. & Gerhart, S. L., Toward a theory of test data selection, *Proc. Int. Conference on Reliable Software*, Los Angeles, april 1975. Ook in: *SIGPLAN Notices*, 10(6), juni 1975, pag. 493-510.
7. Grindley, K., *Systematics - a new approach to systems analysis*, Mc Graw-Hill, London, 1975, 200 pag.
8. Lew, A., Optimal conversion of extended-entry decision tables with general cost criteria, *Communications of the ACM*, 21(4), april 1978, pag. 269-279.
9. Lew, A., On the emulation of flowcharts by decision tables, *Communications of the ACM*, 25(12), december 1982, pag. 895-905.
10. Lew, A., Decision tables for general-purpose scientific programming, *Software - Practice and Experience*, 13, 1983, pag. 181-188.
11. Lew, A., Proof of correctness of decision table programs, *The Computer Journal*, 27(3), 1984, pag. 230-232.
12. Lew, A., *Computer Science: a mathematical introduction*, Prentice-Hall, Englewood-Cliffs, 1985, 421 pag.
13. Lundeberg, M., Goldkuhl, G. & Nilsson, A., *De ISAC-methodiek*, Samsom, Alphen a/d Rijn, 1981, 378 pag.
14. Maes, R., Recente evoluties van de beslissingstabellentechniek: een literatuuroverzicht, *Informatie*, 19(2), februari 1977, pag. 74-78.
15. Maes, R., On the representation of program structures by decision tables: a critical assessment, *The Computer Journal*, 21(4), november 1978, pag. 290-295.
16. Maes, R., An algorithmic approach to the conversion of decision grid charts into compressed decision tables, *Communications of the ACM*, 23(5), mei 1980, pag. 286-293.
17. Maes, R., Bijdrage tot een kritische herwaardering van de beslissingstabellentechniek, K.U.-Leuven, proefschrift, 1981, 397 pag.
18. Maes, R. & Mercken, R., Aspecten van softwaretesten, *Beleidsinformaticatijdschrift*, 7(2), 1981, 53 pag.
19. Maes, R., Vanthienen, J. & Verhelst, M., Procedural decision support through the use of PRODEMO, *Proc. 2nd Int. Conference on Information Systems*, Boston, 1981, pag. 135-153.

20. Maes, R., On minimizing decision grid charts, *Angewandte Informatik*, 24(9), september 1982, pag. 451-455.
21. Maes, R., Beslissingstabellen, in: *Handboek voor Informatieverzorging*, Samsom, Alphen a/d Rijn, pag. 5150/1-16.
22. Maes, R., Vanthienen, J. & Verhelst, M., Practical experiences with the PROcedural DEcision MOdeling system, *Proc. IFIP Working Conference on Processes and Tools for Decision Support*, North-Holland, 1983, pag. 139-154.
23. Maes, R., A composed program complexity measure, *Angewandte Informatik*, 17(1), januari 1985, pag. 9-16.
24. Maes, R. & Van Dijk, J. E. M., A user-friendly propositional formalism for a managerial DSS-generator, *Proc. Int. Workshop on Artificial Intelligence in Economics and Management*, Zürich, 1985, te verschijnen (voorpublikatie als Research Memorandum No. 8516 van de Fac. der Economische Wetenschappen, Universiteit van Amsterdam, 1985, 19 pag.).
25. Martelli, A. & Montanari, U., Optimizing decision trees through heuristically guided search, *Communications of the ACM*, 21(12), december 1978, pag. 1025-1033.
26. Mc Cabe, T. J., A complexity measure, *IEEE Transactions on Software Engineering*, SE-2(4), december 1976, pag. 318-320.
27. Mc Mullen, B., Structured decision tables, *SIGPLAN Notices*, 19(4), april 1984, pag. 34-43.
28. Moret, B. M. E., Thomason, M. G. & Gonzalez, R. C., Optimization criteria for decision trees, Univ. of New Mexico, Dept. of Computer Science, Techn. Report CS81-6, 1981, 14 pag.
29. Myers, G. J., *The Art of Software Testing*, Wiley-Interscience, New York, 1979, 177 pag.
30. Papakonstantinou, G., A recursive algorithm for the optimal conversion of decision tables, *Angewandte Informatik*, 22(9), september 1980, pag. 350-354.
31. Pooch, U. W., Translation of decision tables, *ACM Computing Surveys*, 6(2), juni 1974, pag. 125-151.
32. Reinwald, L. T. & Soland, R. M., Conversion of limited-entry decision tables to optimal computer programs I: minimum average processing time, *Journal of the ACM*, 13(3), juli 1966, pag. 339-368.
33. Schumacher, H. & Sevcik, K. C., The synthetic approach to decision table conversion, *Communications of the ACM*, 19(6), juni 1976, pag. 343-351.
34. Sethi, I. K. & Chatterjee, B., Conversion of decision tables to efficient sequential testing procedures, *Communications of the ACM*, 23(5), mei 1980, pag. 279-285.
35. Vanthienen, J., Automatisering van de beslissingstabellentechniek, K.U.-Leuven, Dept. voor Toegepaste Economische Wetenschappen, niet-gepubliceerde doctoraal seminarietekst, 1985, 80 pag.
36. Verhelst, M., *De praktijk van beslissingstabellen*, Kluwer, Deventer/Antwerpen, 1980, 175 pag.
37. Walsh, D. A., Structured unit testing: a new basis for software quality control, *Software Systems Engineering*, Online, 1976, pag. 341-359.
38. Welland, R., *Decision tables and computer programming*, Heyden & Son, Bury St Edmunds, 1981, 203 pag.

Noten

1. Voor een uitvoerige literatuurlijst: zie (5).

2. Voor een overzicht: zie (14), (31) en (38). Een recenter algoritme is dat van Sethi en Chatterjee (34).

3. Een uitgebreide en tegelijk toegankelijke behandeling van dit onderwerp kan worden gevonden in (29).

4. We zouden dit het 'multiple decision coverage' criterium kunnen noemen.

5. Merk op dat we ons in dit stadium niet bekommeren om het actiegedeelte van de tabel.

6. Een 'limited entry' beslissingstabel bevat, in tegenstelling tot 'extended entry' tabellen, enkel condities die met TRUE (J of Y) en FALSE (N) kunnen worden beantwoord. Myers gebruikt hiervoor de 0- en 1-ingangen.

7. Zie (2) voor een algemene bespreking van dit aspect. Een methodiek die hierbij een vooraanstaande rol vervult betreft de ISAC-methodiek (13).

8. De naam PRODEMO is een afkorting voor het 'PROcedural DEcision MOdeling system'. Een aangepaste, voor micro-computer beschikbare versie van dit systeem betreft het PROLOGA-systeem (zie hiervoor (35)).