DATATYPEN BEZIEN VANUIT DE RECURSIETHEORIE

J.A. BERGSTRA
Rijksuniversiteit Leiden

1. INLEIDING

In deze voordracht willen wij verslag uitbrengen van ons onderzoek van het afgelopen jaar naar de mogelijkheden om met gebruikmaking van het begrippenkader van de recursietheorie inzicht te krijgen in de logische achtergronden van datatypen, datastructuren en implementaties daarvan. De hoofdlijnen kunnen als volgt worden samengevat:

A) Na vervanging van 'functie' door 'proces' als centraal begrip in de (gewone) recursietheorie ontstaat een zogenaamde dynamische recursietheorie (DRT). Deze heeft veel gemeen met de gewone recursietheorie maar staat veel dichter bij 'information processing'.

B) Een natuurlijke subrecursieve variant van deze DRT krijgt men door slechts recursieve (in de zin van berekenbare) processen te beschouwen en processen tot elkaar te herleiden middels eindige automaten. In dit systeem wordt 'proces' een natuurlijke generalisatie van bijv. de Turing-tape. Tot op zekere hoogte kan men stellen: datatype ≡ proces.

C) De identificatie datatype ≡ proces gaat voorbij aan de mogelijkheid van datatypen met een niet-deterministisch karakter. Dit geeft aanleiding tot de definitie: Datatype ≡ drietal $<L_I, L_O, C>$ waarin C een collectie van processen is met inputtaal L_I en outputtaal L_O.

D) 'Datatype' is een 'logisch' begrip. 'Datastructuur' daarentegen niet. In deze opzet wordt 'datastructuur' een per definitie niet precies te omschrijven begrip.

D) DRT vergelijkt geheugenstructuren algoritmisch, op dezelfde wijze als gewone RT informatieverzamelingen algoritmisch vergelijkt.

2. DYNAMISCHE RECURSIE-THEORIE

2.1. Processen

Processen communiceren met de buitenwereld W m.b.v. een inputtaal Σ_I en een outputtaal Σ_O. Proces P is aanvankelijk in toestand P_ε. De mogelijke toestanden van P worden genoteerd met P_σ, $\sigma \in \Sigma_I^*$. Om precies te zijn: P_σ is steeds een functie van $\Sigma_I * \Sigma_I^*$ naar Σ_O. P_ε heet de grafiek van P. $P_\sigma(\tau) = k$ betekent: Zij P in toestand P_σ en laat achtereenvolgens input $(\tau)_1,(\tau)_2 \ldots (\tau)_\ell$ ($\ell = \mathrm{lth}(\tau)$) aan P gegeven worden, dan geeft P als laatste output k. Kennelijk geldt: $P_\sigma(\tau) = P_\varepsilon(\sigma*\tau)$.
We willen nu berekeningen definiëren met processen als argumenten. De 'regel' voor het gebruik van een proces P is als volgt: Zij P in toestand P_σ, geven we P input $a \in \Sigma_I$ dan wordt output $P_\sigma(a)$ gegeven. De nieuwe toestand van P is $P_{\sigma*a}$. Het verschil met het gebruik van een orakel (: $\mathbb{N} \to \mathbb{N}$ of $\mathbb{N} \to \{0,1\}$, of $\Sigma^* \to \{0,1\}$) zit in de toestandsverandering. Bij een proces moet men liever denken aan bijv. de Turing-tape. De collectie van processen (in hun begintoestand) met inputalfabet Σ_I en outputalfabet Σ_O noteren we in het vervolg met $\mathrm{PR}(\Sigma_I,\Sigma_O)$.

2.2. Automaten

Zij $\Sigma_I,\Sigma_O,\Sigma_I^1,\Sigma_O^1 \ldots \Sigma_I^n,\Sigma_O^n$ een n+1 - tal paren, T, van in- en outputalfabetten. Een automaat M van type T is een 3n+5 - tal

$$M = \langle S,s_0,\delta,(IN,OUT),(OUT^1,s^1,IN^1) \ldots (OUT^n,s^n,IN^n)\rangle.$$

Hierbij is:

S de verzameling van toestanden, $s_0 \in S$ de begintoestand
(Zij $K = S \times \Sigma_I \times \Sigma_O^1 \times \ldots \times \Sigma_O^n$),
δ: $K \to S$ de transitiefunctie,
OUT: $K \to \Sigma_O \cup \{NO\}$ de outputfunctie,
IN^i: $K \to \Sigma_I^i \cup \{NO\}$ de inputfunctie voor het i-de proces,
IN het inputregister,
OUT^i het outputregister voor het i-de proces,
s^i de beginwaarde van OUT^i.

NO is een symbool $\notin \Sigma_O \cup \Sigma_I^1 \cup \ldots \cup \Sigma_I^n$. IN kan elementen uit Σ_I bevatten, de OUT^i zijn registers voor Σ_O^i.

Zij $P \in PR(\Sigma_I,\Sigma_O)$, $Q^i \in PR(\Sigma_I^i,\Sigma_O^i)$. Onderstaande beschrijving bevat een informele definitie van

$$P = M(\vec{Q})$$

(M simuleert P m.b.v. $Q^1 \ldots Q^n$).

We gaan uit van de Q^i in hun begintoestand en M in state s_0. Zodra een input a aan het systeem (lees: M) wordt gegeven krijgt IN de waarde a. Het argument $k \in K$ voor δ, OUT en de IN^i is steeds $<s,y,x_1,\ldots,x_n>$ waarbij s de toestand is, y de waarde is van IN en x_i de waarde van OUT^i. M kan nu op de bekende wijze van toestand naar toestand lopen. Wanneer OUT(k) = NO wordt er geen output gegeven, zo ook wordt er, wanneer $IN^i(k)$ = NO geen input aan het proces Q^i gegeven.

Is $OUT(k) = b \in \Sigma_O$ dan wordt b als output gegeven. Feitelijk resulteert dit in een verandering van IN. IN krijgt de waarde van de nieuwe input. Dit alles wordt geacht in een stap te geschieden. Op soortgelijke wijze resulteert $IN^i(k) = b \in \Sigma_I^i$ in een verandering van OUT^i. OUT^i wordt de output van Q^i op input b. (Hierbij verandert Q^i van toestand.)

2.3. $D_{IN}^{\leq}$

2.3.1. Zij PR = PR(IN,IN).

DEFINITIE. $P \leq \vec{Q}$ als er een automaat M is, van het goede type, zodat:

i) $P = M(\vec{Q})$;

ii) S_M is een recursieve deelverzameling van IN.

iii) δ, OUT en de IN^i zijn recursieve functies.

Het is eenvoudig in te zien dat $P \leq Q$ een transitieve relatie levert. We kunnen nu, juist als in de gewone recursietheorie, de door $\leq$ voortgebrachte equivalentierelatie uitdelen.

2.3.2. DEFINITIE. $D_{IN}^{\leq} = <PR/\equiv,\leq,+,0>$.

Hierbij is $\leq$ de op $PR/\equiv$ geïnduceerde partiële ordening. 0 is de laagste graad (bijv. van de constante processen, i.e. de processen welke altijd eenzelfde output geven). Noteren we de eq. klasse van P met d(P) (degree of P) dan is d(P) + d(Q) = d(P+Q) waarbij P + Q als volgt algoritmisch gedefinieerd kan worden:

$$P + Q_{s_1 \ldots s_k}(a) = \begin{cases} \text{als } k \text{ even dan } 0 \\ \text{als } k \text{ oneven en } s_k = 0 \text{ dan } P(a) \\ \text{als } k \text{ oneven en } s_k \neq 0 \text{ dan } Q(a). \end{cases}$$

OPMERKING. $P + Q$ is niet noodzakelijk het supremum van P en Q. Bijv. hoeft $P + P \equiv P$ niet te gelden. Het supremum $P \vee Q$ kunnen we als volgt definiëren:

$$(P \vee Q)_{s^1 * \sigma}(a) = \begin{cases} \text{als } s^1 = 0 \text{ dan } P(a) \\ \text{anders } Q(a). \end{cases}$$

2.3.3. Functionele processen

DEFINITIE. Het proces P heet functioneel wanneer voor zekere functie $f: \mathbb{N} \to \mathbb{N}$ geldt (voor alle σ)

$$P_\sigma(a) = f(a).$$

Notatie voor P: P^f.

FEIT. $P^f \leq P^g \iff f \leq g$ (in de gewone zin).

DEFINITIE. P heet EF ('eventually functional') als voor elke $h: \mathbb{N} \to \mathbb{N}$ er een k is zodat

$$P_{h(1) \ldots h(k)}$$

functioneel is.

FEIT. Stel $f < g$ dan is er een Q met:

i) $P^f < Q < P^g$;

ii) $Q \equiv R$ impliceert R niet EF.

CONCLUSIE. $D^{\leq}_{\mathbb{N}}$ levert een echte verfijning op van de gewone graden van onoplosbaarheid.

DEFINITIE. P is van minimale graad (MIN(P)) als

i) $0 < P$;

ii) $R \leq P \Rightarrow (R \equiv 0 \vee R \equiv P)$.

STELLING. *Er zijn processen van minimale graad.*

2.3.4. Open problemen

1) Hoeveel minimale graden zijn er?

2) Bestaat er een minimale graad beneden elke graad $\neq 0$?

3) Bevat $D^{\leq}_{IN}$ mod(P) voor gegeven P ϵ PR een minimale graad ('minimal cover' van P)?

4) Is $d = d_1 + d_2$ een relatie die uitdrukbaar is in de eerste orde taal van de struktuur $\langle PR/\equiv, \leq, 0\rangle$?

2.3.5. Operatoren

DEFINITIE. Een operator F is een functie: PR $\to$ PR die, gezien als operatie op grafieken, continu is (in de producttopologie op $\mathbb{N} \to \mathbb{N}$, na codering van grafieken in zulke functies).

Operatoren kunnen gebruikt worden door automaten M^O die

i) voorzien zijn van registers voor processen;

ii) expliciete definitie van processen toestaan in de vorm van het volgende assignment:

$$Q_i \Leftarrow F(M_1^O(Q_{i_1} \ldots Q_{i_\ell}, \vec{F})).$$

Hierbij zijn de Q_j processen in het j-de register. Het effect van dit assignment is tweeledig:

- Q_i krijgt de waarde F(P) met $P = M_1^O(\vec{Q}_{i_k}, \vec{F})$,
- $Q_{i_1} \ldots Q_{i_\ell}$ krijgen de waarde $\lambda\sigma \cdot 0$ (deze processen zijn a.h.w. verbruikt).

Het is nu duidelijk hoe de relatie

$$P = M^O(\vec{Q}, \vec{F})$$

kan worden gedefinieerd.

Tenslotte definiëren we

$$F \leq \vec{G}, \vec{Q} \text{ voor operatoren } F, \vec{G}$$

door:

$$\exists M^O \; \forall P \; F(P) = M^O(P, \vec{Q}, \vec{G}).$$

Dit leidt tot een gradenstructuur op de collectie van operatoren.

We kunnen twee soorten van operatoren onderscheiden:

- operatoren recursief in een proces.
- operatoren niet recursief in enig proces.

Van de tweede categorie is de hieronder beschreven operator IN een (universeel) voorbeeld.

IN(P) wordt bepaald door de volgende eigenschappen:

$$\begin{cases} IN(P)_{\sigma * 0} = IN(P)_{\varepsilon} \\ IN(P)_{s_1+1,\ldots,s_k+1}(s+1) = P_{s_1,\ldots,s_k}(s). \end{cases}$$

(IN(P) wordt door 0 geïnitialiseerd, IN(P) $\upharpoonright$ $(IN-\{0\})^* \equiv P$).

PROBLEMEN.

5) Zijn er operatoren van minimale graad?
6) Zit er tussen twee proces-graden een operator-graad?
7) Hoeveel operator-graden zijn er?

2.4. D_{Σ}^{FC} .

2.4.1. Een subrecursieve variant van $D_{IN}^{\leq}$. Zij Σ een aftelbaar oneindig alafbet.

DEFINITIE. RP_{Σ} is de collectie van processen P met input-taal Σ_I^P, output-taal Σ_O^P zodat:

i) Σ_I^P, Σ_O^P eindig deel van Σ
ii) na codering van Σ_I^P en Σ_O^P in IN is P recursief in de zin van 2.3.1 (P≡0).

De gedachte is dat voor de praktijk interessante processen uit PR zich in RP_{Σ} bevinden. (gerechtvaardigd zolang er geen nondeterminisme in het spel is).

DEFINITIE. Zij P, $\vec{Q}$ in RP_{Σ}

$$P \leq_{FC} \vec{Q}$$

als $P = M(\vec{Q})$ geldt voor een automaat M met eindig veel toestanden. (P is 'finite conrol reducible to' $\vec{Q}$).

2.4.2. DEFINITIE. $D_{\Sigma}^{FC} = \langle RP_{\Sigma}/\equiv_{FC}, \leq_{FC}, +, 0\rangle$.

OPMERKINGEN.

i) + en $\vee$ gedragen zich als bij $D_{IN}^{\leq}$;

ii) operatoren kunnen op soortgelijke wijze behandeld worden;

iii) functionele processen hebben nu graad 0. Hun rol wordt overgenomen door de initialiseerbare processen (die in $D_{IN}^{\leq}$ alle functioneel zijn).

DEFINITIES.

INIT(P): P is initialiseerbaar $\equiv$ voor zekere $a \in \Sigma_I^P$ geldt $\forall \sigma \in (\Sigma_I^P - \{a\})^* \; P_{\sigma * a} = P_{\varepsilon}$.

INIT(d): graad d bevat een initialiseerbaar proces.

MIN(d) : d is minimale graad ($\neq 0$).

MIN(P) : P is van minimale graad.

MON(P) : Er bestaat een ordening $\preccurlyeq$ op Σ_O^P zodat voor elke $\sigma, a, b: P_{\sigma}(a) \preccurlyeq P_{\sigma * a}(b)$ (P heet monotoon).

MON(d) : d bevat monotoon proces.

L(P) : P is lineair $\equiv \Sigma_I^P$ bevat één element.

L(d) : d bevat een lineair proces.

RED(P) : P is recudibel $\equiv$ er zijn Q en R met $Q < P$, $R < P$ en $P \equiv Q + R$.

2.4.3. RESULTATEN t.a.v. D_{Σ}^{FC}.

STELLING 1.

i) *Er is een maximale graad* M;

ii) *Bij elk proces is er een equivalent proces met twee inputsymbolen en twee outputsymbolen.*

BEWIJS (aanduiding).

i) M is de graad van de Turing-tape TT. TT beschrijven we informeel als volgt:

Σ_I = {READ, WRITE-0, WRITE-1, MOVE-LEFT, MOVE-RIGHT}

Σ_O = {0,1,BL,BLANK}

TT gedraagt zich als een Turing-tape die aanvankelijk geheel met BL gevuld is. Op READ komen antwoorden 0, 1 of BL. Op de andere instructies (inputs) volgt antwoord BLANK. Vanzelfsprekend is elk recursief proces met behulp van TT en 'finite control' te simuleren.

ii) Codeer input- en outputsymbolen in binaire strings.

STELLING 2. *Er is een proces van minimale graad.*

BEWIJS (aanduiding). Zij

Σ_I = {UP,DOWN,BOTTOM}

Σ_O = {0,1}

$$A_\sigma(k) = \begin{cases} 0 & \text{als } k = \text{BOTTOM en } \sigma = UP^t * DOWN^t \text{ voor zekere } t \in \mathbb{N} \\ 1 & \text{anders.} \end{cases}$$

Met enige moeite bewijst men nu:

i) $A \not\equiv_{FC} 0$;

ii) $B \leq_{FC} A^n \wedge B \not\equiv_{FC} 0 \Rightarrow A \leq_{FC} 0$.

De gedachte achter ii) is als volgt. Stel B voldoet aan de condities, B = M(A). Dan verschilt B op oneindig veel plaatsen van $M(P_1)$, P_1 het constant 1 proces. Dus zijn er willekeurig lange runs σ waarop $M(A)(\sigma)$ van $M(P_1)(\sigma)$ verschilt. Dit kunnen we nu gebruiken om een run te construeren waarbij vijf fasen te onderscheiden zijn: aanloop/repeterend gedeelte/gedeelte waarin A de eerste DOWN instructie krijgt/volgend repeterend gedeelte/uitloopgedeelte waarin A een signaal (0) geeft dat op e.o.a. wijze door M(A) wordt doorgegeven.

Gegeven deze situatie is het niet moeilijk meer om $A \leq B$ aan te tonen.

STELLING 3.

i) TT *is niet van lineaire graad;*

ii) *er is een lineair proces* L *zodat* $TT \equiv L + L + L + L$.

BEWIJS.

i) Op betrekkelijk eenvoudige wijze is aan te tonen dat het predicaat $P_M^L(s,k) \iff$ 'M in state s stopt op $L_0 k$' beslisbaar is. Dit zou niet mogelijk zijn wanneer TT m.b.v. L simuleerbaar is. (Hier is L een willekeurig lineair proces.)

ii) Neem L als volgt: $\Sigma_I = \{0\}$, $\Sigma_O = \{0,1\}$. L geeft eerst een 0, dan een 1, dan weer een 0, dan twee 1-en, dan weer een 0, vervolgens drie 1-en enz.
Met enig werk blijkt het mogelijk te zijn om een counter, CO, m.b.v. L + L te simuleren. Daar CO + CO ≡ TT geldt TT ≡ (L+L) + (L+L).

STELLING 4.

i) *Een minimale graad is monotoon;*

ii) *Een initialiseerbare graad* > 0 *is niet monotoon;*

iii) *Een lineaire graad* > 0 *is niet monotoon;*

iv) *De som van monotone graden is weer monotoon.*

CORROLARIUM. M *is niet de som van (eindig veel) minimale graden.*

BEWIJSSCHETS.

i) Zij $P \not\equiv 0$. Er is een uniforme constructie voor $Q \leq P$ zodat $Q \not\equiv 0$ en Q monotoon.

ii)...iv) Berusten er op dat monotone processen tijdens een run slechts eindig veel keren niet-triviale informatie kunnen geven.

2.4.4. Open problemen (t.a.v. D_Σ^{FC})

8) Hoeveel minimale processen zijn er?

9) Is er onder elke graad $\not\equiv 0$ een minimale graad?

10) heeft elke graad $\not\equiv$ TT een 'minimal cover'?

11) Is er een paar graden A en B zonder infimum?

12) Is de elementaire theorie van D_Σ^{FC} beslisbaar?

13) Bestaan er niet-minimale irreducibele graden?

14) (Indien JA op 13) Is elke graad de som van eindig veel irreducibele graden? (Hoe is het met M in het bijzonder?)

2.5. D_{Σ}^{EFC}

Het is mogelijk om met een variant $\leq_{EFC}$, (extend finite control), van $\leq_{FC}$ processen met oneindige in- en outputalfabetten tot elkaar te herleiden. Hiertoe voert men een eindig aantal extra registers voor symbolen in. Het gebruik ervan is beperkt tot:

i) een registerinhoud als OUT-put of IN^i-put geven;

ii) inhouden van twee registers op gelijkheid testen;

iii) assignment van de inhoud van IN en de OUT^i aan de nieuwe registers.

Zij f nu een vaste bijectie: $\mathbb{N} \to \Sigma$; RP'_{Σ} is de collectie van processen P met:

i) Σ_I^P, Σ_O^P zijn recursieve deelverzamelingen van Σ;

ii) P is recursief.

(In i) en ii) moet men recursief lezen m.b.t. de identificatie van Σ en $\mathbb{N}$ midels f.)

FEIT 1. $P,Q \in RP_{\Sigma}$, $P \leq_{EFC} Q \Rightarrow P \leq_{FC} Q$.

FEIT 2. $P \in RP_{\Sigma}$, $Q \in RP'_{\Sigma}$, $Q \leq_{EFC} P \Rightarrow \exists R \in RP_{\Sigma}\ Q \equiv_{EFC} R$.

Uit deze feiten concluderen wij dat $\leq_{EFC}$ een zeer natuurlijke uitbreiding van $\leq_{FC}$ naar RP'_{Σ} is.

3. DATATYPEN

Duidelijk is dat vele datatypen behorende bij abstracte machinemodellen passen in D_{Σ}^{FC}. Dit deel is gewijd aan enkele opmerkingen over de beschrijving van andere datatypen.

3.1. Datatypen uit de complexiteitstheorie

Zij Λ een vast eindig alfabet. Een 'probleem' is een deelverzameling A van Λ^*.

Voor problemen A en B is het bekend (COOK) wat men moet verstaan onder: A is in polynomiale tijd reduceerbaar tot B (in de zin van Turing-reduceerbaar met B als orakel). Notatie $A \leq_T^P B$.

We willen dit nu in termen van processen beschrijven. Allereerst moeten we bij A,B passende processen $\tilde{A},\tilde{B}$ vinden.

DEFINITIE. $\tilde{A}$. $\Sigma_I^{\tilde{A}} = \Delta \cup \{\#\}$

$\Sigma_O^{\tilde{A}} = \{0,1,BL\}$.

Voorts:

$$\tilde{A}_{\sigma*\#} = \tilde{A}_{\varepsilon},\ \tilde{A}_{\sigma}(k) = BL \text{ als } k \neq \#$$

$$\tilde{A}_{\substack{\sigma \\ \sigma\in\Delta^*}}(\#) = \begin{cases} 0 & \text{als } \sigma \in A \\ 1 & \text{anders.} \end{cases}$$

Informeel gesproken: $\tilde{A}$ is een initialiseerbare karakteristieke functie van A.

STELLING. *Er bestaat een proces* P *in* RP_Σ *met de volgende eigenschap:*

$$A \leq_T^P B \iff \tilde{A} \leq_{FC} \tilde{B},\ P.$$

BEWIJSSCHETS. We geven een summiere beschrijving van P. P leest eerst een rij $\sigma \in \{0,1\}^*$ in die de exponent exp bepaalt, gevolgd door een #. Daarna is P in toestand $P_{\sigma_{exp}}$. Vervolgens leest P weer een string $\tau \in \{0,1\}^*$ gevolgd door #. Deze string staat nu op een tape waarover een leeskop kan heen- en weerlopen en lezen. Zij ℓ de lengte van τ. Nu gedraagt P zich als een Turing-machine met twee tapes, de zojuist beschreven leestape met het input-woord τ, en een gewone lees- en schrijftape, oorspronkelijk overal gevuld met BL. Echter, wanneer meer dan ℓ^{exp} instructies zijn uitgevoerd levert P nog slechts output BLANK. Wordt # ingelezen dan keert P terug naar toestand $P_{\sigma_{exp}}$.

OPMERKING. Een soortgelijk resultaat is te vinden voor LOGSPACE reducibility.

PROBLEEM.

15) Bestaat er P zodat $\tilde{A} \leq_{FC} \tilde{B},P \iff \tilde{A}$ reduceerbaar tot $\tilde{B}$ in lineaire (kwadratische) tijd?

3.2. Algebraïsche datatypen

Beschouw de algebraïsche structuur $\langle \mathbb{N}, SUC, 0, = \rangle$. We willen er relevante processen bij definiëren, analoog aan het effect van de ALGOL-60 declaratie integer i,j,k; Deze declaratie roept in onze situatie een proces P in het leven met (informeel genoteerd)

$$\Sigma_I^P = \{i \Leftarrow 0,\ j \Leftarrow 0,\ k \Leftarrow 0,\ i \Leftarrow SUC(i),\ j \Leftarrow SUC(j),\ k \Leftarrow SUC(k),\ i = j?\}$$
$$\Sigma_O^P = \{0,1,BL\}.$$

Hierbij komt 0 of 1 als output op i = j? inputs en BL als output in de andere gevallen. Het is duidelijk hoe P zich dient te gedragen.

Voorts is duidelijk dat de overgang van een algebraïsche structuur naar een bijbehorend proces van heel wat parameters afhangt. Binnen D_Σ^{FC} kunnen de resultaten van verschillende keuzen kwalitatief met elkaar worden vergeleken.

3.3. Niet-deterministische datatypen

De identificatie datatype ≡ proces houdt geen rekening met niet-deterministische verschijnselen. Vandaar onderstaande algemene definitie.

DEFINITIE. Een datatype is een drietal

$$D = \langle \Sigma_I, \Sigma_O, C_D \rangle,$$

waarbij $C_D \subseteq PR(\Sigma_I, \Sigma_O)$ niet leeg. (Meestal bevat C_D slechts recursieve processen.)

D heet deterministisch als C_D een singleton is. De elementen van C_D zijn implementaties van D.

DEFINITIE. D_1 is relatief implementeerbaar t.o.v. D_2 ($D_1 \leq D_2$) wanneer er een (extended) finite control machine $M_{(E)FC}$ is zodat geldt:

$$\forall P \in C_{D_2}[M_{(E)FC}(P) \in C_{D_1}].$$

OPMERKING. Kiest men de Σ_I, Σ_O weer deel van een vaste Σ dan genereert ≤ op de zo verkregen collectie van niet-deterministische datatypen een structuur soortgelijk aan de Medvedev lattice [1].

Het is niet moeilijk om voorbeelden D_1 en D_2 te vinden met D_1 deterministisch en D_2 niet-deterministisch zodat $D_1 \leq D_2$ geldt.
Bijvoorbeeld: D_2 registreert eindige deelverzamelingen van een gegeven verzameling E en heeft ter beschikking de operaties INSERT(V,e), DELETE(V,e), EMPTY(V)? en de niet-deterministische instructie $x_E \Leftarrow SEL(V)$ die tot gevolg heeft dat x_E de waarde krijgt van enig element in V (mits V niet leeg). Voor D_1 kan men nu nemen: weer de eindige deelverzamelingen

van E ditmaal zonder SEL maar met UNION en INTERSECTION. Deze laatste twee kunnen m.b.v. de in D_2 beschikbare operaties worden gerealiseerd op uniforme wijze (i.e. onafhankelijk van de preciese implementatie van SEL).

PROBLEEM.

16) Geef een uitwerking van bovenstaand voorbeeld (o.i.d.) met: een axiomatische beschrijving van D_1 en van D_2 en een correctheidsbewijs van de (relatieve) implementatie.

4. AUTONOME PROCESSEN

In een rekenmachine kan een functie TIME gerealiseerd worden. Deze hangt van 'de tijd' af, meer dan van de inputhistorie. Er is alle reden om TIME als een (autonoom) datatype op te vatten. In elk geval kunnen we TIME zien als een autonoom proces. Deze sectie is gewijd aan het precies invoeren van autonome processen en recursie er op. Intuïtief gesproken krijgt een autonoom proces zo nu en dan een input en geeft het zo nu en dan een output. Wanneer we nu gediscretiseerde tijd $t_0,t_1,t_2,t_3,\ldots$ aannemen dan is het echter natuurlijk om te veronderstellen dat een AP elk tijdstip input krijgt en output geeft. We kunnen een speciaal symbool BLANK reserveren om lege (i.e. niet bedoelde) in- en output aan te duiden. Het blijkt dat PR en RP_Σ nu als collecties van autonome (recursieve autonome) processen kunnen worden opgevat. We zullen ze noteren met APR en ARP_Σ. Op verrassend eenvoudige wijze kunnen we reduceerbaarheid op APR en ARP_Σ definiëren. Namelijk door de definitie van automaat M zodanig te wijzigen dat de mogelijkheden OUT(k) = NO en $IN^i(k)$ = NO komen te vervallen (of, wat hetzelfde is, NO te vervangen door BLANK). Bovenstaande geeft aanleiding tot het vormen van de structuren

$$AD_\omega^{\leq} = \langle ARP/\equiv,\leq,+,0\rangle$$
$$AD_\Sigma^{FC} = \langle ARP_\Sigma/\equiv_{FC},\leq_{FC},+,0\rangle.$$

Het is onze overtuiging dat zich binnen $AD_\omega^{\leq}$ en AD_Σ^{FC} belangwekkende zaken afspelen.

PROBLEMEN.

17) Structuur van $AD_\omega^{\leq}$.

18) Structuur van AD_Σ^{FC}.

19) Vind natuurlijke inbeddingen van D_Σ^{FC} in AD_Σ^{FC}.

20) Vind binnen AD_Σ^{FC} geheugenmechanismen welke intrinsiek van autonome aard zijn.

LITERATUUR

[1] HARTLEY ROGERS Jr., *The theory of recursive functions and effective computability*, McGraw Hill.