Over de sequentialiteit van procesbeschrijvingen.

Het is niet ongebruikelijk, wanneer een spreker zijn voordracht begint met een inleiding. Omdat ik mij hier mogelijk richt tot een gehoor, dat als geheel minder vertrouwd is met de problematiek, die ik wil aansnijden en de terminologie, die ik zal moeten gebruiken, wilde ik in dit geval ter introductie graag twee inleidingen houden, nl. een om de achtergrond van de problemen te schilderen en een tweede, om U een gevoel te geven voor de aard van de logische problemen, die wij tegen het lijf zullen lopen.

Mijn eerste inleiding is een historische. Eerst moet ik U vertellen, wat U zich moet voorstellen bij een sequentiele procesbeschrijving. U kent allen dergelijke procesbeschrijvingen. Beschouw bv. de beschrijving van de constructie van de hoogtelijn uit het hoekpunt C van de driehoek ABC. Deze kan als volgt luiden:

1) trek de cirkel met middelpunt A en straal = AC;
2) trek de cirkel met middelpunt B en straal = BC;
3) trek de rechte, die bepaald is door de snijpunten van de onder 1) en 2) genoemde cirkels.

Dit is een beschrijving ten bate van iemand, wiens "parate kennis" de constructie van de hoogtelijn niet omvat; ten bate van hem is de constructie van de hoogtelijn opgebouwd uit een aantal handelingen van een beperkter repertoire, handelingen, die hij in de opgegeven volgorde, de een na de ander, dwz. sequentieel, kan uitvoeren. Het zal het oplettende lezertje niet ontgaan zijn, dat wij hierbij meer gespecificeerd hebben, dan strikt noodzakelijk: het is wel essentieel, dat handeling 3) pas uitgevoerd wordt, nadat handelingen 1) en 2) voltooid zijn, handelingen 1) en 2) hadden best in de omgekeerde volgorde plaats mogen vinden, sterker, een tekenaar, die twee passers tegelijkertijd bedienen kan, bij wijze van spreken met elke hand één, mag handelingen 1) en 2) simultaan uitvoeren. Maar deze vrijheid is in onze procesbeschrijving geheel onder tafel geraakt.

Nu een numerieker voorbeeld, dat iets meer aansluit bij de denkwereld, waarin de straks te behandelen problemen actueel zijn geworden. Gegeven de waarden van de variabelen a, b, c en d, gevraagd te berekenen de waarde van de variabele x, bepaald door de formule "x:=(a+b)*(c+d)". Wanneer het uitvoeren van deze waardetoekenning niet tot het primitieve repertoire van de rekenaar -en denkt U nu langzamerhand maar aan de rekenmachine- behoort, dan dient dit rekenvoorschrift uit elementairdere stappen opgebouwd te worden, bv.:

1) bereken t1:= a + b;

2) bereken t2:= c + d;

3) bereken x:= t1 * t2.

Dit rekenvoorschrift bevat precies dezelfde overbepaaldheid als onze hoogtelijn-constructie: de volgorde van de eerste twee stappen doet niet ter zake, zij zouden zelfs simultaan uitgevoerd kunnen worden.

De bovenstaande procesbeschrijving is representatief voor de werkingswijze van meer klassieke rekenautomaten. Het repertoire van elementaire operaties is, omdat voor elke operatie speciale technische voorzieningen vereist zijn, eindig en om financiele redenen zelfs heel beperkt; deze machines danken hun enorme flexibiliteit aan het feit, dat willekeurig ingewikkelde algebraische expressies als boven geillustreerd uitgewerkt kunnen worden door een sequens van dergelijke elementaire operaties. Inhaerent aan deze sequentiele beschrijving van het proces is, dat men meer specificeert, dan strikt noodzakelijk. Dit heeft lange tijd niet gehinderd, maar in de laatste jaren is daar verandering in gekomen, en wel door twee oorzaken: verandering van de structuur van machines en de komst van een nieuwe klasse problemen, waarvoor men deze machines zou willen inzetten.

De klassieke sequentiele machine doet "een ding tegelijk" en verricht de ene functie na de andere. Om der wille van de efficiency is men voor specifieke functies meer specifieke apparatuur gaan bouwen, maar als je er dan aan vast houdt, dat de verschillende operaties strict sequentieel uitgevoerd zullen worden, dan dringt zich van een machine plotseling het beeld op van een samenspel van N organen, waarvan er op elk discreet ogenblik maar 1 of 2 werken. Als N groot is, betekent dit, dat het merendeel van je machine het merendeel van de tijd stilstaat en dat is kennelijk niet de bedoeling.

De tweede oorzaak is de komst van een nieuwe categorie van problemen; oorspronkelijk werden rekenautomaten alleen gebruikt voor van te voren volledig gedefinieerde processen, en de machine kon het proces vervolgens op zijn eigen houtje, in zijn eigen tempo uitvoeren. Zodra men echter het informatieverwerkende proces samen wil laten spelen met een of ander ander proces -bv. een chemisch proces, dat door de rekenautomaat gestuurd moet worden- dan treden er heel andere problemen op. Aanhakend bij ons voorbeeld van de berekening van x:=(a + b)*(c + d) zijn we dan niet meer in de omstandigheid, dat het er niet meer toe doet, welke som het eerste gevormd wordt. Als de gegevens a,b,c en d op onbekende momenten door de omgeving van de rekenautomaat verschaft worden en het de taak van de machine is, om de waarde van x zo snel mogelijk af te leveren, dan betekent dit, dat de auto-

maat een deelberekening maar vast uit moet voeren, zodra de daarvoor vereiste gegevens binnen zijn: welke optelling men dan eerst uitgevoerd wil hebben, zal afhangen van de volgorde. waarin de gegevens a, b, c en d ter beschikking komen. Dit was mijn eerste inleiding en ik hoop U hiermee een idee gegeven te hebben, van wat U zich bij een sequentiele procesbeschrijving voor moet stellen, en waarom de vraag gesteld is naar minder strict sequentiele prcesbeschrijvingen.

Nu mijn tweede inleiding, die bedoeld is om U van meet af aan enig idee te geven van de logische problemen, die we door het verlaten van de stricte sequentialiteit ons op de hals zullen halen.

Evenals in een normaal stuk algebra variabelen met letters, algemeen "namen", aangeduid worden, heeft men in het proces van informatieverwerking aanhoudend de plicht tot identificatie, de plicht om bij de naam het hierdoor benoemde object te vinden. Onder omstandigheden zou men graag willen beschikken over het selectieproces, waarover de juffrouw in de klas beschikt, wanneer zij zegt "Jantje, kom eens voor het bord." Aangenomen, dat de klas als geheel oplet en gehoorzaam is, werkt dit proces feilloos onder de aanname, dat er 1 en slechts 1 Jantje in de klas zit. Het bijzondere van dit selectiemechanisme is, dat het werkt ongeacht het aantal kinderen in de klas. Deze idylle is helaas technisch niet of hoogstens gebrekkig te verwezenlijken, en en wij krijgen een probleemstelling, die meer in overeenstemming is met de werkelijkheid, wanneer de kinderen niet spontaan op het noemen van hun naam reageren en de juffrouw, wanneer zij Jantje voor het bord wil hebben, genoodzaakt is om Jantje bij het oor te vatten en daaraan voor het bord te slepen. Als dit nu een ouderwetse school is, waarin ieder kind zijn vaste plaats in de klas heeft, dan kan de juffrouw, als zij aan het begin van het jaar de kinderen op hun plaats gezet heeft, ze voor haar eigen gemak herdopen, en de naam "Jantje" -die voor het vinden van het jongetje niet erg behulpzaam is- vervangen door een identificatie, die blijkens zijn structuur meteen definieert, welk oor ze vatten moet: ze kan het jongetje noemen "derde rij, tweede bank". Dit is in wezen de techniek, die in klassieke automaten bij klassieke toepassingen gevolgd wordt. In modernere toepassingen kan men dit niet meer doen, en moet men bv. opgewassen zijn tegen de problemen, die de juffrouw krijgt, als de kinderen kunnen kiezen, waar ze gaan zitten.

Om het de juffrouw niet onmogelijk te maken, nemen we aan dat elk der onwillige kinderen een naamplaatje opgespeld heeft. Als de juffrouw nu Jantje voor het bord wil hebben, moet zij dus Jantje opzoeken. De juffrouw werkt sequentieel, kan

slechts een naamplaatje tegelijkertijd lezen, en de juffrouw werkt systematisch: zij heeft voor zichzelf de banken op een of andere manier uniek geordend en als zij Jantje wil hebben, dan loopt zij de banken in deze volgorde af, zich steeds afvragend "Ben jij Jantje?", zo nee, dan gaat ze naar de volgende bank. Dit proces werkt feilloos, mits wij kunnen garanderen, dat de kinderen tijdens de rondgang van de juffrouw niet gaan verzitten, mits wij kunnen garanderen, dat de twee processen "kindlocalisatie" en "kinderpermutatie" niet simultaan uitgevoerd worden. Want U begrijpt ook wel, dat Jantje, zodra hij door heeft, dat hij de gezochte figuur is, achter in de klas gaat zitten en zodra de juffrouw de voorste helft van de klas doorzocht heeft, springt hij gauw naar voren. Wij kunnen de juffrouw verfijnen en haar, als ze de hele klas doorzocht heeft en Jantje niet gevonden heeft met frisse moed van voren af aan laten beginnen, maar dit is geen oplossing, want Jantje springt dan gauw weer naar achteren. Nu zou U kunnen opmerken, dat als die permutaties nu niet met zoveel kwaadwilligheid, maar slechts random uitgevoerd werden, dat dan de verwachtingstijd voor het localiseren van Jantje eindig blijft, ja zelfs door zijn vrijheid rond te springen niet groter is geworden. Deze opmerking, hoe hoopvol, brengt geen soulaas. Zij die zich verdiept hebben in de "Theory of computability" weten hoeveel gewicht gehecht wordt aan een eindige, dwz. een gegarandeerd eindigende algorithme, en zij zullen begrijpen, dat een proces, dat potentieel oneindig lang doortolt, geen aantrekkelijk element is. Veel erger is, dat het in de praktijk veelal de vraag is, of wij de permutaties van de kinderen wel als random mogen beschouwen. Als de kans eindig is, dat als de juffrouw een keer door de klas geweest is, de kinderen weer precies zo zitten als toen de juffrouw begon te zoeken, dan kon dit best eens de eerste periode van een zuiver periodiek verschijnsel zijn.

Maar zelfs als wij het risico, dat we Jantje nooit zullen vinden voor lief nemen, dan nog is haar ellende niet te overzien, want naast het risico, dat ze Jantje niet vindt, loopt ze het veel grotere gevaar, dat ze het verkeerde kind voor het bord sleurt, nl. als de kinderen tussen het moment, dat zij voldaan geconstateert heeft "Zo, jij ben dus Jantje" en het moment, dat zij, op grond van deze constatering haar hand naar het bijbehorend oor heeft uitgestoken, nog gauw van plaats verwisselen. In een informatieverwerkend proces zou dit neerkomen op een ramp. En dit is het einde van mijn tweede inleiding, waarvan ik hoop, dat zij U enig idee gegeven heeft van het soort probleem, dat wij tegen zullen komen.

Aan het einde van mijn eerste inleiding heb ik twee oorzaken genoemd, die voor een groeiende belangstelling in minder strict sequentiele procesbeschrijvingen

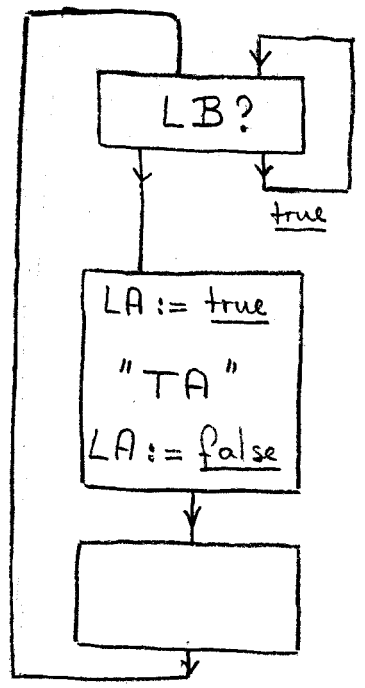
verantwoordelijk waren. Ik heb ze slechts luchtig aangestipt en het is U misschien niet opgevallen, dat ze aan bijna tegengestelde entourages ontleend waren. In het eerste geval noemde ik de toenemende complexiteit van machines, die nu opgebouwd zijn uit een aantal min of meer autonoom werkende onderdelen en de opgave was, om deze samen één proces uit te laten voeren. In het tweede geval, waar we de automaat in een z.g. "real time application" bekeken, werd het beeld opgeroepen van één enkel apparaat, dat de mogelijkheid had bij een verandering van de externe urgentie-situatie over te schakelen op het nu urgenste van zijn mogelijke taken. Deze twee arrangementen, een samenspel van een aantal machines aan één proces en omgekeerd, één machine zijn aandacht verdelend over een aantal processen, zijn lange tijd als elkaar wezensvreemd beschouwd. Men heeft er zelfs verschillende namen voor uitgevonden, het ene noemt men "parallel programming" en het andere noemt men "multiprogramming" maar U moet mij niet meer vragen wat wat, want dat kan ik niet meer onthouden, sinds ik ontdekt heb dat de logische problemen, die zij door de non-sequentialiteit van de procesdefinitie oproepen, in beide gevallen exact dezelfde zijn.

Ik zal mij in het volgende bedienen van het beeld van een aantal, onderling zwak gekoppelde, opzichzelf sequentiele machines, daarbij aansluitend op de eerste entourage. Maar dit is slechts een beschrijvingswijze, want met wat ik in het vervolg een machine zal noemen komt niet noodzakelijkerwijze een discreet aanwijsbaar stuk apparatuur overeen: in het volgende zijn mijn onderscheiden machines mogelijk heel abstract en representeren zij niet meer dan een opzichzelf sequentieel deelproces. Dit komt onder andere daarin tot uiting, dat ik een grote voorliefde zal tonen voor een samenspel van N machines, N willekeurig. Als ik de term machine alleen maar bezigde voor een aanwijsbaar functioneel stuk apparatuur, dan zou ik het aantal machines voor elke installatie als vrij constant mogen beschouwen. Nu echter een machine , als personifiering van een opzichzelf sequentiele deeltaak, door de gebruiker ad libitum gecreeerd kan worden, is een dergelijke vaste bovengrens niet meer acceptabel. Ik ga dus praten over het samenspel van N machines, N niet alleen willekeurig, maar als het moet zelfs tijdens het proces variabel. Overtollige machines kunnen vernietigd worden, nieuwe machines kunnen naar behoefte er bij gecreeerd worden en in het samenspel der overigen worden opgenomen.

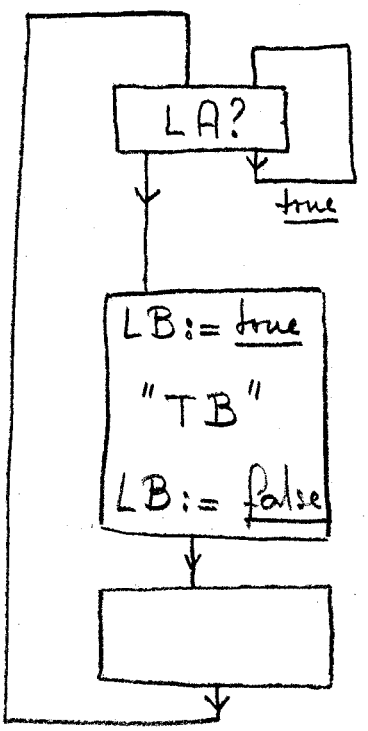
Ik hoop, dat U thans op mijn gezag wilt aannemen, danwel na afloop zelf inziet, dat wij ons zonder de algemeenheid te schaden kunnen beperken tot sequentiele machines, die een of ander cyclisch proces uitvoeren.

Laat ons beginnen met een heel eenvoudig probleem. Gegeven twee machines A en B, beide bezig met een cyclisch proces. In de cyclus van machine A komt een zeker critisch traject voor, TA genaamd, en in die van machine B komt een critisch traject TB voor. De opgave is om er voor te zorgen, dat nooit gelijktijdig de beide machines elk aan hun critische traject bezig zijn. (In termen van onze tweede inleiding: machine A zou kunnen zijn de juffrouw, traject TA het selectieproces van een nieuwe leerling, die voor het bord gehaald moet worden, terwijl het traject TB het proces "kinderpermutatie" representeert.) Er mogen geen veronderstellingen gemaakt worden over de relatieve snelheden der machines A en B, de snelheid, waarmee zij werken hoeft zelfs niet constant te zijn. Het is duidelijk, dat wij de wederzijdse uitsluiting van het critische traject slechts kunnen realiseren, als de twee machines op een of andere manier met elkaar kunnen communiceren. Voor deze communicatie stellen wij enig gemeenschappelijk geheugen ter beschikking, dwz. een aantal variabelen, die voor beide machines in beide richtingen toegankelijk zijn, dwz. waaraan beide machines een waarde kunnen toekennen en waarvan beide machines naar de heersende waarde kunnen informeren. Deze twee handelingen, toekennen van een nieuwe waarde en informeren naar de heersende waarde gelden als ondeelbare handelingen, dwz. als beide machines "tegelijkertijd" een waarde aan een gemeenschappelijke variabele willen toekennen, dan is de waarde na afloop of de ene, of de andere toegekende waarde, maar niet een of ander mengsel. Evenzo: als de ene machine informeert naar de waarde van een gemeenschappelijke variabele op het moment, dat de andere machine er een nieuwe waarde aan toekent, dan dringt tot de vragende machine of de oude, of de nieuwe, maar niet een wilde waarde door. Het zal blijken, dat we ons voor dit doel kunnen beperken tot gemeenschappelijke logische variabelen, dwz. variabelen met slechts twee mogelijke waarden, die we in aansluiting op standaard technieken met "true" respectievelijk "false" aangeven. Voorts nemen wij aan, dat beide machines slechts naar 1 gemeenschappelijke variabele tegelijkertijd kunnen refereren, en dan ook alleen of om een nieuwe waarde toe te kennen of om naar de heersende waarde te informeren. In figuur 1 is in blokschemavorm een tentatieve oplossing gegeven voor de structuur van de programma's voor beide machines. Elk blok heeft zijn ingang boven en zijn uitgang onder. Bij een blok met een dubbele uitgang wordt de keuze bepaald door de waarde van de zojuist aangevraagde logische variabele: heeft deze de waarde true, dan is de uitgang rechtsonder bedoeld, heeft deze de waarde false, dan kiezen we de uitgang linksonder.

Er komen in dit schema twee gemeenschappelijke logische variabelen voor, LA en LB. LA betekent: machine A is in zijn critische sectie, LB betekent: machine B

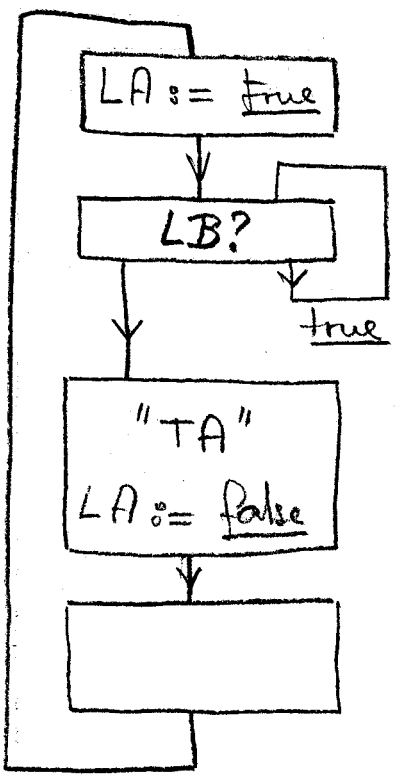


machine A



machine B

Fig 1



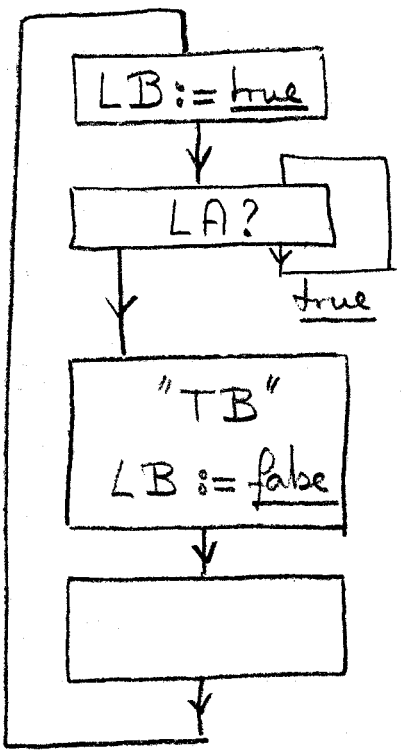


Fig 2.

is in zijn critische sectie. De schema's in fig 1 zijn duidelijk. In het blok bovenaan wacht machine A indien hij daar aankomt op een ogenblik, dat machine B in sectie TB bezig is, totdat machine B zijn critische traject heeft verlaten, wat door de toekenning "LB:= false" gemarkeerd wordt, waardoor dan de wacht van machine A wordt opgeheven. En omgekeerd. De schema's mogen dan eenvoudig zijn, ze zijn helaas ook fout, want ze zijn wat te optimistisch: ze sluiten nl. niet uit, dat beide machines tegelijkertijd in hun respectievelijke critische trajecten terecht komen. Als beide machines buiten hun critische traject zijn -zeg ergens in het blanco gelaten blok- dan zijn zowel LA als LB false. Komen ze nu tegelijkertijd in hun bovenste blok, dan vinden ze beide, dat de andere machine hen geen strobreed in de weg legt, en ze gaan beide door en komen tegelijkertijd in hun critische sectie.

We zijn dus wat te optimistisch geweest. De fout is geweest, dat de ene machine niet wist, of de ander al naar zijn staat van vordering informeerde. De schema's van fig.2 maken een veel betrouwbaardere indruk en het is gemakkelijk te verifieren, dat zij volslagen veilig zijn. Bv. machine A kan slechts aan traject TA beginnen, nadat tijdens het true-zijn van LA geconstateerd is, dat LB false is, en ongeacht hoe snel na deze kennis van machine A dit feit weer obsoleet wordt, doordat machine B gauw zijn bovenste blok "LB:= true" uitvoert, machine A kan dan veilig traject TA ingaan, omdat machine B toch netjes opgehouden wordt totdat LA na afloop van traject TA op false gezet wordt. Deze oplossing is dus volkomen veilig, maar we moeten niet denken, dat we hiermee het probleem opgelost hebben, want deze oplossing is te veilig, er bestaat nl. de kans, dat het hele samenspel van deze machines vastloopt. Als zij tegelijkertijd het bovenste blok van hun schema uitvoeren, dan lopen beide machines in de daarop volgende wacht en blijven tot in eeuwigheid van dagen beleefd voor dezelfde deur staan, zeggende "Na U" ,"Na U". We zijn te pessimistisch geweest.

U ziet, dat het probleem niet triviaal is. Het was mij althans, na deze twee probeersels , helemaal niet zonneklaar, dat er een veilige oplossing bestond, die niet tevens de mogelijkheid van een doodlopende weg in zich hield. Ik heb het probleem toen in deze vorm aan mijn toenmalige collegae van de Rekenafdeling van het Mathematisch Centrum voorgelegd, er bij vertellend, dat ik niet wist, of het oplosbaar was. Aan Dr.T.J.Dekker komt de eer toe als eerste een oplossing gevonden te hebben, die bovendien symmetrisch in de beide machines was. Deze oplossing is weergegeven in fig.3. Er is een derde logische variabele ingevoerd, nl. AP, die betekent, dat in geval van twijfel machine A prioriteit heeft. Door de asymmetrische

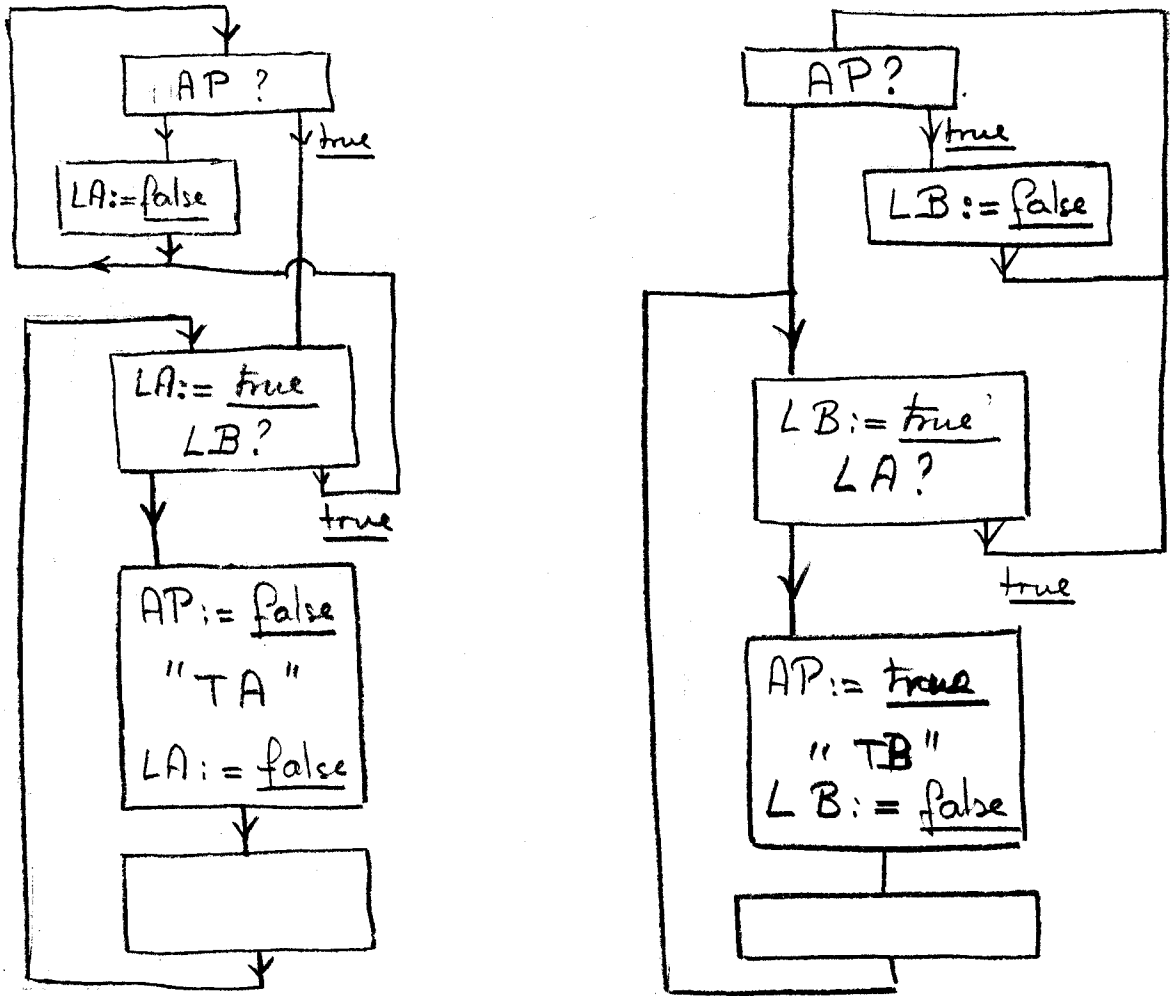


Fig 3.

betekenis van deze logische variabele zijn gedeelten van deze programma's min of meer elkaars spiegelbeeld, functioneel is de oplossing echter helemaal symmetrisch en is niet de ene machine bevoorrecht boven de ander. Om te bewijzen, dat deze oplossing aan de gestelde eisen voldoet, constateren we eerst, dat de trajecten TA en TB voorafgegaan worden door de sluis, die we in figuur 2 al als veilig herkend hebben. Simultane uitvoering is dus onmogelijk. We hoeven slechts te verifieren, dat het nu bovendien uitgesloten is, dat beide machines voor hun critische sectie blijven wachten. In deze wachtcycli wordt de waarde van AP niet gewijzigd. Beperken we ons tot het geval, dat AP true is, dan kan machine A slechts blijven cyclen als ook LB true is; onder de voorwaarde AP true kan machine B echter alleen rond blijven lopen in de cyclus, die LB op false zet, dwz. machine A uit zijn cyclus forceert. In het geval AP false verloppt de analyse op dezelfde manier. Omdat AP als logische variabele of true of false is, is daarmee de mogelijkheid van eeuwig op elkaar blijven wachten eveneens uitgesloten. Q.E.D.

Dekker's oplossing dateert uit 1959 en bijna drie jaar heeft deze oplossing voortgeleefd als "curiositeit", totdat dit soort problematiek aan het begin van 1962 voor mij plotseling weer actueel werd en Dekker's oplossing als uitgangspunt van mijn volgende pogingen gefungeerd heeft.

De probleemstelling was inmiddels wel drastisch veranderd. In 1959 was het de vraag of de beschreven communicatiemogelijkheid tussen twee machines het mogelijk maakte om twee machines zodanig te koppelen, dat de uitvoeringen van de critische trajecten elkaar in de tijd wederzijds uitsloten. In 1962 werd een veel wijdere groep problemen bekeken en werd bovendien de vraagstelling veranderd in "Welke communicatiemogelijkheden tussen machines zijn gewenst, om dit soort spelletjes zo sierlijk mogelijk te spelen?"

Dekker's oplossing heeft op verschillende manieren als uitgangspunt gefungeerd. Omdat het in deze oplossing onduidelijk is, hoe men te werk moet gaan, wanneer het aantal machines uitgebreid wordt, en de opgave blijft om te garanderen, dat er op elk moment hoogstens 1 critische sectie onder behandeling is, was zij een aansporing om naar andere wegen te zoeken. Door een analyse van de moeilijkheden, die Dekker moest overwinnen, konden wij een aanwijzing krijgen wat een hoopvollere weg zou zijn om in te slaan.

De moeilijkheden ontstaan, doordat als een van de machines geinformeerd heeft naar de waarde van een gemeenschappelijke logische variabele, deze informatie onmiddellijk daarna al weer obsoleet kan zijn, nog voordat de informerende machine er effectief op gereageerd heeft, in het bijzonder: nog voordat de informerende machine aan zijn partners heeft kunnen meedelen, dat hij van de waarde van een gemeenschappelijke variabele "een momentopname gemaakt heeft". De hint, die uit deze bespiegeling gehaald kan worden, is dat men moet zoeken naar machtigere "elementaire" operaties op de gemeenschappelijke variabelen, in het bijzonder naar operaties, waarin voor de informatietransport tussen machine en gemeenschappelijk geheugen niet meer de beperking van het éénrichtingverkeer geldt.

De meestbelovende operatie, die we hebben kunnen bedenken, is de operatie "falsify". Deze informeert naar de waarde van een logische en het feit, dat deze operatie is uitgevoerd wordt in de helft van de gevallen kenbaar gemaakt: de operatie falsify laat de logische variabele, waarop hij geopereerd heeft nl. met de waarde "<u>false</u>" achter.

De operatie "falsify" kan als volgt gedefinieerd worden:

"boolean procedure falsify(S); boolean S;
begin boolean Q; falsify:= Q:= S; if Q then S:= false end" ,

mits we hierbij vermelden, dat de operatie falsify, in weerwil van zijn sequentiele definitie, beschouwd moet worden als elementaire opdracht van het repertoire der machines, waarbij "elementair" in dit geval betekent, dat gedurende de operatie falsify door één van de machines de logische variabele in kwestie ontoegankelijk is voor alle andere machines.

Dat deze operatie een stap in de goede richting is, volgt wel uit het feit, dat we nu meteen de oplossing hebben voor een wolkje machines X1, X2, X3,....., elk met zijn critisch traject TXi, die elkaar nu alle in de tijd moeten uitsluiten. We kunnen dit spelen met 1 gemeenschappelijke logische variabele, zeg SX, die aangeeft, dat geen van de machines in zijn critische traject bezig is. De programma's vertonen nu alle dezelfde structuur:

"LXi: if non falsify(SX) then goto LXi; TXi; SX:= true; proces Xi; goto LXi"

waarbij we de aandacht er op vestigen, dat in de programma's voor de aparte machines Xi nergens tot uiting komt, hoeveel machines Xi er eigenlijk zijn: we kunnen dus door loutere toevoeging het wolkje machines Xi uitbreiden.

Het geprogrammeerde wachtcyclusje, dat hierin voorkomt, is natuurlijk heel aardig, maar het beantwoordt wat weinig aan ons doel. Een wachtcyclusje is immers de manier om een machine "zonder effect" bezig te houden. Maar als we nu bedenken, dat het merendeel van deze machines door een centrale computer gesimuleerd zal worden, zodat elke actie in de ene machine slechts plaats kan vinden ten koste van de effectieve snelheid van de andere machines, dan is het wel erg begrotelijk om in een wachtcyclusje aandacht van de centrale computer op te gaan eisen voor iets volslagen nutteloos. Zolang het wachtcyclusje niet verlaten kan worden, mag wat ons betreft de snelheid van deze wachtende machine tot nul zakken, Om dit tot uitdrukking te brengen voeren we in plaats van het wachtcyclusje een statement in, een elementaire opdracht van het repertoire van de machines, die heel heel lang kan duren. We geven dit aan met een P (van Passering); vooruitlopend op latere behoeften representeren we de statement "SX:= true" door "V(SX) -met de V van Vrijgave. (Deze terminologie is ontleend aan het spoorwegwezen: in een eerder stadium heetten de gemeenschappelijke logische variabelen "Seinpalen" en als hun naam met een S begint, dan is dat nog een reminiscentie daaraan.) De text van de programma's luidt in deze nieuwe notatie:

"LXi: P(SX); TXi; V(SX); proces Xi; goto LXi" .

De operatie P is een tentatieve passering. Het is een operatie, die slechts beeindigd kan worden op een moment, dat de er bij opgegeven logische variabele de waarde true heeft, beeindiging van de operatie P houdt tevens in, dat aan de logische variabele in kwestie weer de waarde false wordt toegekend. Operaties P kunnen wel simultaan plaatsvinden, het beeindigen van een P-operatie wordt echter weer beschouwd als een ondeelbare gebeurtenis.

De invoering van de operatie V is meer dan een verkorte schrijfwijze invoeren. Immers: toen wij het wachtcyclusje met behulp van de operatie falsify nog hadden, rustte de detectieplicht ten aanzien van het gelijk true worden van de gemeenschappelijke logische variabelen op de individuele machines, die op deze gebeurtenis stonden te wachten en in die zin was het wachten een tamelijk actieve bezigheid op een betrekkelijk neutrale gebeurtenis. Met de invoering van de P-operatie hebben wij zo ongeveer bereikt, dat wij tegen een machine, die voorlopig niet verder kan, zeggen: "Ga maar slapen, we maken je wel weer wakker, als je verder mag." Maar dit betekent, dat het op true zetten van een gemeenschappelijke logische variabele, waarop misschien een machine staat te wachten, niet meer een neutrale gebeurtenis is als ieder andere waardetoekenning: hier kan nl. het neveneffect aan verbonden moeten worden, dat een van de "slapende" machines "gewekt" moet worden. Het betekent, dat de logische variabelen, die gedurende false-zijn één of meer machines op kunnen houden, niet permanent geobserveerd hoeven te worden, mits bij overgang naar de waarde true een "centrale wekker" op deze overgang attent gemaakt kan worden. Als de centrale wekker per gemeenschappelijke logische variabele een administratie bij houdt, welke machines op elke logische variabele op het ogenblik wachten, dan kan de centrale wekker, mits deze expliciet op het true worden van een dergelijke gemeenschappelijke logische variabele geattendeerd wordt, snel decideren, of er op grond van deze overgang een slapende machine gewekt kan worden en zo ja welke. De operatie V kan nu opgevat worden als een combinatie van het toekennen van de waarde true aan een seinpaal, plus het attenderen van de centrale wekker op deze gebeurtenis. Het si duidelijk, dat deze centrale wekker zijn werk niet kan doen, als we toestaan, dat de gemeenschappelijke logische variabelen ook nog, door normale waardetoekenningen (van het neutrale type "S:= true"), "stiekem", ondershands gewijzigd kunnen worden. Om dit verbod te onderstrepen, noem ik de gemeenschappelijke logische variabelen van nu af aan "seinpalen" en stel ik, dat de enige mogelijkheid, waarop een machine met een bestaande seinpaal zal kunnen communiceren, zal zijn de P-operatie en de V-operatie.

En hiermede is de operatie "falsify", die ons door de aantrekkelijkheid van zijn eenvoud bij de begripsvorming zo behulpzaam is geweest, weer van het toneel verdwenen en de vraag is gerechtvaardigd, of we door de introductie van de seinpaal, die slechts via de operaties P en V toegankelijk is, het kind niet met het badwater hebben weggegooid. Dit is gelukkig niet het geval. Door de operatie "falsify" van het elementaire repertoire van de onderscheiden machines af te voeren en daar de operaties P en V voor in de plaats te geven, hebben wij niets verloren: wil de invoering van de operaties P en V niet een lege geste zijn, dan moet er dus minstens 1 seinpaal zijn, waarop zij kunnen werken. Deze ene seinpaal- noem hem "SG", de Seinpaal Generaal- is echter al voldoende om de operatie falsify op een willekeurige gemeenschappelijke logische variabele toe te voegen aan het repertoire van alle machines. Immers:

Als wij in elke machine definieren:
"boolean procedure falsify(GB); boolean GB;
begin boolean Q; P(SG); falsify:= Q:= GB; if Q then GB:= false; V(SG) end"
en wij voorts afspreken, dat elke referentie naar een gemeenschappelijke logische variabele in elke machine ingekapseld zal worden tussen de statements "P(SG)" en "V(SG)" (zodat de assignment "GB1:= true" nu de vorm zal hebben "P(SG); GB1:= true; V(SG)", dan hebben wij hiermee verzekerd dat aan de bijvoorwaarde voldaan is, dat " gedurende de operatie falsify door één van de machines de logische variabele in kwestie ontoegankelijk is voor alle andere machines." We hebben met de ruil dus niet in eigen vlees gesneden.

Wij gaan nu laten zien, dat de operatie's P en V in hun toepassingsgebied niet beperkt zijn tot wederzijdse uitsluiting, maar dat we ze ook kunnen gebruiken, om twee machines ten opzichte van elkaar te synchroniseren. Stel, dat wij twee cyclische machines hebben, zeg X en Y, die elk in hun cyclus 'n critische sectie hebben, zeg TX resp. TY, waarbij de volgende eisen gesteld zijn:

1) de uitvoering van de critische sectie TX mag in de tijd niet samenvallen met die van TY;
2) de uitvoeringen van de secties TX en TY moeten alternerende gebeurtenissen zijn (op deze voorwaarde doelden we, toen we spraken van "onderlinge synchronisatie").

Als U zich een entourage wilt voorstellen, waarin men voor dit probleem gesteld wordt, denkt U dan maar aan het volgende. Proces X is een cijferproducerend proces (een rekenproces), proces Y is een cijferconsumerend proces (een schrijfmachine) en deze twee processen zijn gekoppeld door een "toonbank" met de capaciteit van één cijfer. De handeling TX is het neerleggen van het volgende te typen

cijfer op een lege toonbank, het stuk TY representeert hét afnemen van het cijfer van de volle toonbank. De stricte alternering vloeit voort uit de eis dat enerzijds geen enkel aan de toonbank aangeboden cijfer "weg mag raken" maar inderdaad getypt moet worden, en dat anderzijds -als er een tijdje geen aanbod is- de schrijfmachine niet zijn tijd moet gaan vullen met het nog maar eens typen van het laatste cijfer.

Wij kunnen dit bereiken met twee seinpalen, zeg SX en SY, waarbij:
SX betekent, dat nu eerst een sectie TX aan de beurt is, en
SY betekent, dat er nu eerst een sectie TY aan de beurt is.
Voor machines X en Y luiden de programma's nu:

"LX: P(SX); TX; V(SY); proces X; goto LX" en
"LY: P(SY); TY; V(SX); proces Y; goto LY" .

Wij kunnen dit direct op twee verschillende manieren uitbreiden. Indien wij de machine X vervangen door een wolkje machines Xi, die allemaal van dezelfde toonbank gebruik willen maken om hun productie te lozen, en de machine Y vervangen door een wolkje machines, die allemaal bereid zijn, om informatie van de toonbank af te nemen (ongeacht, door welke van de machines Xi het er op is gelegd, het beeld van de schrijfmachines gaat hier dus niet meer op), dan luiden de programma's voor machines Xi en Yj analoog aan de boven gegeven programma's:

"LXi: P(SX); TXi; V(SY); proces Xi; goto LXi" en
"LYj: P(SY); TYj; V(SX); proces Yj; goto LYj" .

Hadden we ook nog een groepje machines Zk gehad, en was de opgave geweest, dat de uitvoering van een TX-sectie, een TY-sectie en een TZ-sectie elkaar in die volgorde cyclisch moesten opvolgen, dan hadden de programma's geluid:

"LXi: P(SX); TXi; V(SY); proces Xi; goto LXi" en
"LYj: P(SY); TYj; V(SZ); proces Yj; goto LYj" en
"LZk: P(SZ); TZk; V(SX); proces Zk; goto LZk" .

Wij merken hierbij op, dat in de laatste twee voorbeeldjes de machines Xi gelijk zijn: zij representeren uitsluitend, dat de handelingen TX ergens door geinterspatieerd moeten worden, zij laten in het midden, hoe complex deze tussenliggende handelingen gerealiseerd zullen worden.

Een volgende opgave moge aantonen, dat onze operatie's P en V in hun huidige vorm nog niet helemaal machtig genoeg zijn. Wij beschouwen daartoe de volgende configuratie. We hebben twee machines A en B, elk met hun critische sectie TA respectieve-

lijk TB; deze secties zijn volledig onafhankelijk van elkaar, maar er is nog een derde machine C in het spel met zijn critische sectie TC en de uitvoering van TC mag in de tijd niet samen vallen met die van TA, noch met die van TB. We kunnen dit doen met twee seinpalen, zeg SA en SB. Voor machines A en B is het programma eenvoudig, nl.

"LA: P(SA); TA; V(SA); proces A; goto LA" en

"LB: P(SB); TB; V(SB); proces B; goto LB" ,

maar voor machine C is het volgende programma. hoewel veilig, onacceptabel:

"LC: P(SA); P(SB); TC; V(SB); V(SA); proces C; goto LC" .

Immers: machine C zou de statement P(SA) succesvol voltooid kunnen hebben op een moment, dat machine B in sectie TB bezig is. Zolang sectie TB nu nog niet voltooid is, kan machine C niet verder -dat is correct- maar ook machine A kan zijn sectie TA niet meer uitvoeren en dat is niet correct: indirect, via machine C zijn hier de secties TA en TB toch weer gekoppeld. Door de operatie P(SA) van machine C is de seinpaal SA overhaast op false gezet, daarmee machine A onnodig hinderend.

. Wij gaan daarom de operaties P en V uitbreiden tot operaties, die we een willekeurig aantal argumenten mee kunnen geven -daarmee de grenzen van ALGOL 60 overschrijdend, maar dat doet er in dit verband bitter weinig toe. Voor de V-operatie is de betekenis duidelijk: het is niet anders dan de simultaneous assignment, die aan alle meegegeven seinpalen gelijktijdig de waarde true toekent. De P-operatie met een aantal argumenten, bv. "P(S1, S2, S3)", is een operatie, die slechts beeindigd kan worden op een ogenblik, dat alle meegegeven seinpalen de waarde true hebben (in ons voorbeeld dus een ogenblik, waarop geldt: "S1 and S2 and S3"), beeindiging van een P-operatie houdt dan in, dat aan alle meegegeven seinpalen simultaan de waarde false toegekend wordt. Ook na deze uitbreiding wordt de beeindiging van een P-operatie weer beschouwd als een ondeelbare gebeurtenis.

Met deze uitbreiding zijn we in staat machine C zodanig te definieren, dat het gesignaleerde bezwaar niet meer optreedt:

"LC: P(SA, SB); TC; V(SA, SB); proces C; goto LC".

Tenslotte wil ik, hoewel dit beslist niet de toepassing is, waarvoor de P- en V-operaties geconcipieerd zijn, laten zien, hoe we hiermee het scalair product van twee vectoren kunnen berekenen, zonder dat we ons vast leggen op de volgorde, waarin de producten der elementen bij elkaar opgeteld worden.

Stel, dat we uit willen rekenen de som SIGMA(i,1,5,A[i] * B[i]) -in woorden: de som, voori lopend van 1 t/m 5, van (A[i] * B[i])-;

stel, dat ingevoerd zijn de 7 seinpalen: Ssom, Sklaar en de vijf seinpalen Sterm[i] met i=1,2,3,4 en 5; alle 7 met de beginwaarde false;

stel, dat inmiddels gecreeerd zijn vijf machines van de structuur (i=1,2,3,4,5):

```
"Lterm[i]: P(Ssom, Sterm[i]);
           scapro:= scapro + A[i] * B[i];
           n:= n - 1;
           if n = 0 then V(Sklaar) else V(Ssom);
           goto Lterm[i] " ,
```

alle bezig aan hun -enige- P-operatie;

Beschouw nu een zesde machine, die onder bovengenoemde voorwaarden juist zal beginnen aan de volgende statements:

```
".....; n:= 5;
       scapro:= 0;
       V(Ssom, Sterm[1], Sterm[2], Sterm[3], Sterm[4], Sterm[5]);
       "programma, dat genoemde seinpalen, scalairen en vectorelementen ongemoeid
        laat";
        P(Sklaar);...."
```

Als de laatste P-operatie van de zesde machine voltooid is, heeft n de waarde 0, scapro heeft de waarde van de gevraagde som en alle seinpalen en de vijf eerstgenoemde machines zijn terug in hun oorspronkelijke toestand.