

0. Inleiding.

Om te beginnen beschouwen wij een aantal "zwak gekoppelde", onderling asynchrone, in zich zelf sequentiele processen. Zonder schade te doen aan de algemeenheid kunnen we ons tot cyclische processen beperken.

De zwakke koppeling zal daaruit bestaan, dat deze processen informatie met elkaar kunnen uitwisselen, dwz. toegang hebben tot enig gemeenschappelijk geheugen. Wanneer deze processen samen van een zelfde faciliteit bij toerbeurt gebruik zouden willen maken, of wanneer het ene proces informatie wil consumeren, die door een ander proces geproduceerd moet worden, dan volgen hieruit zekere synchronisatie-eisen. We moeten zorgen, dat processen elkaar niet storen of, in het geval van informatieuitwisseling, niet te ver ten opzicht van elkaar uit de pas geraken.

In het eerste gedeelte wordt in de opzet volledig in het midden gelaten, door welke apparatuur deze onderscheiden processen gerealiseerd worden. De uiteindelijke bedoeling is, om in aansluiting met de gegeven -en ik moet er aan toe voegen: concipieerbare- werkelijkheid sommige van deze cyclische processen door een specifiek stuk hardware te laten uitvoeren, terwijl de overige processen door de centrale computer verzorgd zullen worden. In het eerste gedeelte is dit onderscheid niet van belang; wel belangrijk is, dat een "cyclisch proces" een geprogrammeerde conceptie kan zijn en dat het aantal cyclische processen, dat op elk gegeven moment in het spel betrokken is, in de tijd kan variëren.

In 1. zal ik in de plaats van "cyclisch proces" spreken van "machine", ter onderscheiding streping van het feit, dat we ons deze processen als grotendeels (of volslagen) autonoom moesten voorstellen. In het licht van de vorige alinea moeten we ons er dus niet over verbazen, als er een paar machines bijgeschapen worden of er een paar machines geannihileerd worden. Later, als we dichterbij de realisatie met behulp van beperkte hardware komen, dan zullen we onderscheid maken tussen "concrete machines" -dwz. cyclische processen, waaraan een specifiek stuk hardware beantwoordt zoals bv. een communicatieapparaat- en "abstracte machines", die tezamen door de centrale computer gesimuleerd worden. Voor deze simulator generaal zal ik het woord computer reserveren. Deze simulatie roept, met het oog op efficiënter gebruik van de totale installatie, strategieproblemen op, waarvan ik me op dit ogenblik met nadruk distancieer. Voordat ik nl. het strategieprobleem überhaupt wil bekijken, dwz. voordat ik er ook maar over denken wil, hoe de computer zijn vrijheid zou kunnen uitbuiten, wil ik eerst zijn "onvrijheid" geformaliseerd zien, wil ik een passend formalisme hebben, waarin de logisch noodzakelijke "interlock's" tussen de diverse machines geformuleerd kunnen worden.

In 2. zullen wij een begin maken met de analyse van de simulatie door één computer van de verzamelde abstracte machines. Hier zien wij overigens heel duidelijk, dat over de onderlinge snelheidsverhoudingen der verschillende machines geen veronderstellingen gemaakt mogen worden: als een van de abstracte machines werkt, staan de andere stil (werken de anderen oneindiglangzaam).

1. Seinpalen

De onderlinge temporele restricties worden vastgelegd in zg. "seinpalen". Een seinpaal is een integer, die voor minstens twee verschillende machines toegankelijk is; de mogelijkheden, waarop iedere individuele machine met de seinpalen kan communiceren, is echter beperkt tot een tweetal, de operatie's P en V.

1.1. De operatie V ("Verhoog").

De operatie V kan betrekking hebben op een willekeurig aantal verschillende seinpalen, dus bv. "V(S1,S2,S3)". Als deze operatie in een van de machines voorkomt

-in ons voorbeeld moeten dan S1, S2 en S3 voor deze machine toegankelijke seinpalen zijn- dan is het effect, dat alle opgegeven seinpalen in één ondeelbare handeling met 1 verhoogd worden.

### 1.2. De operatie P ("Prolaag").

De operatie P kan betrekking hebben op een willekeurig aantal seinpalen, dus bv "P(S1,S2,S3)". Als deze operatie in een van de machines voorkomt -in ons voorbeeld moeten dan S1,S2 en S3 voor deze machine toegankelijk zijn- dan begint een operatie, die slechts beëindigd kan worden op een moment, dat alle opgegeven seinpalen positief zijn. Beëindiging van een P-operatie impliceert, dat alle opgegeven seinpalen met 1 verlaagd worden en deze beëindiging geldt als één ondeelbare handeling. Ook voor de operatie P beperken we ons tot het geval, dat de er bij opgegeven seinpalen verschillend zijn.

### 1.3. Enige voorbeelden van het gebruik van seinpalen.

Opm. Vele seinpalen nemen slechts de waarden 0 en 1 aan. In dat geval fungeert de operatie V als "baanvak vrijgeven"; de operatie P -een tentatieve passering- kan slechts voltooid worden, als de betrokken seinpaal (of seinpalen) op veilig staat en passering impliceert dan een op onveilig zetten. De terminologie is ten duidelijkste ontleend aan het spoorwegwezen - althans aan mijn voorstelling daarvan.

1.3.1. Als wij een klasje machines  $X_i$  hebben ( $X_0, X_1, X_2, \dots$ ) en in elke machine komt een kritische sectie  $TX_i$  voor, kritisch in die zin, dat geen twee kritische secties tegelijkertijd onder behandeling mogen zijn, dan kunnen we dit bereiken met een seinpaal, zeg  $SX$ , die in dit simpele geval slechts tweewaardig zal zijn:

$SX = 0$  zal betekenen: een van de machines  $X_i$  is aan zijn kritische sectie bezig

$SX = 1$  zal betekenen: geen van de machines  $X_i$  is aan zijn kritische sectie bezig.

De omschrijving van alle machines is gelijkloidend:

" $LX_i: P(SX); TX_i; V(SX);$  rest proces  $X_i; \text{goto } LX_i$ ".

Als we alle machines op hun gelabelde punt ("aan het begin van de regel") zouden starten met de beginwaarde  $SX = 2$ , dan zouden we verwezenlijkt hebben, dat ongeacht de omvang van het klasje machines  $X_i$  er nooit meer dan twee simultaan aan hun kritische secties bezig zouden zijn. Dit is kennelijk een generalisatie van de probleemstelling van wederzijdse uitsluiting. (Het is precies de situatie van bv. n tape decks aan twee kanalen.)

Wij vestigen er de aandacht op, dat de formulering der individuele machines  $X_i$  onafhankelijk is van de omvang van het klasje machines  $X_i$ , iets wat met het oog op de dynamische variatie van deze omvang wel hoogst gewenst is.

1.3.2. Nu beschouwen we een groepje machines  $X_i$  en een groepje machines  $Y_j$ , elk met hun kritische sectie  $TX_i$  resp.  $TY_j$ . Uitvoering van een van de kritische secties dient uitvoering van alle andere kritische secties uit te sluiten, maar tevens eisen we, dat de uitvoering van een  $TX$ -sectie en een  $TY$ -sectie alternerende gebeurtenissen zijn.

We kunnen dit bereiken met twee tweewaardige seinpalen, zeg  $SX$  en  $SY$ .

$SX = 1$  betekent dat nu eerst een  $TX$ -sectie aan de beurt is

$SY = 1$  betekent dat nu eerst een  $TY$ -sectie aan de beurt is.

De programma's voor de machines luiden:

" $LX_i: P(SX); TX_i; V(SY);$  proces  $X_i; \text{goto } LX_i$ " en

" $LY_j: P(SY); TY_j; V(SX);$  proces  $Y_j; \text{goto } LY_j$ ".

Als de machines alle op het begin van de regel gestart worden moet  $SX = 1$  en  $SY = 0$  zijn of andersom.

Opm. Hadden we ook nog een groepje machines  $Z_k$  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"

"LYj: P(SY); TYj; V(SZ); proces Yj; goto LYj"

"LZk: P(SZ); TZk; V(SX); proces Zk; goto LZk" .

Wij vestigen er de aandacht op, dat in deze uitbreiding de tekst van de machines Xi niet is veranderd.

1.3.3. Nu beschouwen we een klasje machines Xi, die informatie-eenheden willen lozen in een cyclische buffer met een capaciteit van N informatie-eenheden; voorts een klasje machines Yj, die informatie-eenheden uit deze buffer willen verwerken.

Omdat vullen van de buffer administratieve maatregelen met de "vulwijzer" impliceert -en voor legen mutatis mutandis hetzelfde geldt- eisen we bovendien, dat vullen slechts door 1 machine Xi tegelijkertijd kan geschieden en evenzo, dat legen slechts door een machine Yj tegelijkertijd kan geschieden.

We voeren hiervoor in vier seinpalen:

SX1 = aantal vrije plaatsen in de buffer (aanvankelijk = N)

SX2 = 0 als een van de machines Xi vult, anders 1

SY1 = aantal gevulde plaatsen in de buffer (aanvankelijk = 0)

SY2 = 0 als een van de machines Yj leegt, anders 1.

Er zal gelden  $N - 1 \leq SX1 + SY1 \leq N$ . De machines zijn nu:

"LXi: P(SX1, SX2); vul volgende plaats van de buffer; V(SY1, SY2); proces Xi; goto LXi"

"LYj: P(SY1, SY2); leeg volgende plaats van de buffer; V(SX1, SX2); proces Yj; goto LYj"

In het voorafgaande is het verschil tussen abstracte en concrete machines helemaal op de achtergrond gebleven, juist om te laten zien, dat ze, van een zeker macroscopisch standpunt bezien, volslagen gelijkwaardig zijn. Wij gaan nu dit onderscheid wel maken om te analyseren met wat voor soort hardware dit gerealiseerd zou kunnen worden.

## 2. Onderscheid tussen abstracte en concrete machines.

Een abstracte machine is een of ander geprogrammeerd sequentieel proces, een concrete machine is een transputapparaat. (Transput is input of output.)

Wij interesseren ons vanzelfsprekend speciaal voor het bespelen van seinpalen die de synchronisatie tussen de computer -d.w.z. het aggregaat van abstracte machines- enerzijds en een transputmachine anderzijds regelen.

Voor elke seinpaal van dit type is een vaste geheugenplaats gereserveerd. Deze toevoeging komt op twee manieren tot uiting: enerzijds in de manier, waarop het transputapparaat is aangesloten, anderzijds in het programma, dat de samenwerking met dit transputapparaat verzorgt. Beide processen moeten immers tot deze gemeenschappelijke seinpaal toegang hebben.

Wij beperken ons nu tot twee soorten seinpalen, nl. de seinpaal, waarop de P-operatie door de computer en de V-operatie door de transputmachine wordt uitgevoerd en de seinpaal, waarbij dit juist andersom is.

De afbeelding van de seinpaal in een geheugenplaats ligt heel erg voor de hand: in het algemeen is het een integer, die onthouden moet worden. Als de seinpaal in het

geheugen staat, zijn er vanzelfsprekende opdrachten, waarmee de computer toegang tot de seinpaal kan krijgen. Bovendien zou de transputmachine -als alle- in elk geval al autonoom toegang hebben tot het geheugen. Het basismechanisme voor de seinpaalbespeling is er dus helemaal. Ten verder voordeel van een geheugenplaats is de vanzelfsprekende mogelijkheid tot uitbreiding van het aantal transputmachines (en bijbehorende seinpalen) gegeven. Er is echter 1 complicatie, waarbij we wel even stil mogen staan: van de waarde van ~~XXX~~ een seinpaal -preciezer: van het positief zijn of niet- moet een sturende invloed uitgaan. We kunnen dus niet volstaan met de seinpaal op kernen op te bergen verder niets meer. Aan al deze seinpalen wordt daarom een flip-flop toegevoegd, die een derivaat van de seinpaal zal zijn, nl. =1 als de seinpaal positief is en anders =0. (Zolang we ons beperken tot de ~~XXXXXX~~ non-negatieve seinpalen is de enige andere mogelijkheid =0. We zullen omdat het hier hardware faciliteiten betreft ons geen nodeloze beperkingen opleggen en in onze definities steeds ook negatieve seinpalen toelaten, dwz. we voeren teken-tests uit en geen nul-tests.)

Ten dergelijke toegevoegde flip-flop lost dan wel het probleem op hoe we aan een seinpaal op kernen een sturende invloed kunnen geven, het confronteert ons echter wel met de taak te zorgen, dat als de seinpaal in het geheugen gewijzigd wordt, deze flip-flop zodanig aan de nieuwe toestand wordt aangepast.

Hiervoor zijn twee mogelijkheden

- a) we bevorderen de combinatie van seinpaalwijziging en flip-flop-aanpassing tot de status van ondeelbare handeling.
- b) we gaan bij de separate aanpassing van de flip-flop aan de nieuwe seinpaalwaarde zo omzichtig te werk, dat tijdelijke discrepantie geen brokken kan opleveren.

Waar het transputapparaat een seinpaal wijzigt- kiezen we de eerste methode, waar de computer een seinpaal wijzigt kiezen we de tweede methode. De eerste methode is voor de transputmachine niet moeilijk te verwezenlijken, want de transputmachine heeft immers de mogelijkheid om het geheugen op te eisen -althans met een grotere prioriteit dan de computer. Voor de ~~XX~~ machine is de eerste oplossing minder leuk, want de computer kan immers al bij de seinpalen via normale opdrachten. De automatische aanpassing van de flip-flop zou dus betekenen, dat er op de adresbits een extra uitcodering zou moeten komen; dit ~~XX~~ is niet zo erg aantrekkelijk als we bedenken, dat dit aantal seinpalen toeneemt als er meer apparatuur wordt aangesloten. Verkozen is daarom de methode, waarbij de machine met aparte -matig geadresseerde-opdrachten op elk van de individuele toegevoegde flip-flops kan opereren.

### 2.1. De P-operatie door de transput-machine.

Zij  $t$  de integer in het kerngeheugen, die de seinpaal representeert, zij  $T$  de bijbehorende flip-flop, die =1 is, als  $0 < t$  is.  $T$  fungeert dan als "Actie-flip-flop" voor de communicatiemachine, die op een passend ogenblik in zijn cyclus zal uitvoeren:

"if  $T = 1$  then begin  $t := t - 1$ ; if  $t \leq 0$  then  $T := 0$  end" .

De P-operatie door de transputmachine impliceert bij voltooiing het accepteren van een startopdracht. (Over het wachten als  $T = 0$  laat ik me hier niet uit, bv. of het apparaat werkelijk stil staat of een dode slag maakt, want dat is in dit verband onbelangrijk. Essentieel is, dat de transputmachine bij  $T = 0$  de seinpaal  $t$  en de flip-flop  $T$  ongemoeid zal laten.)

Deze P-operatie -onderdeel van het consumeren van de volgende startopdracht- heeft als mogelijk gevolg, dat het opnemen van volgende startopdrachten voorlopig via de assignment  $T := 0$  verder wordt tegengehouden, nl. als de voorraad is uitgeput.

Additionele spelregels zijn:

- a) dat deze handeling vanuit de computer gezien moet worden als ondeelbare handeling

-dit scheidt zijn voordelen- maar dat anderzijds de computer niet over de mogelijkheid zal beschikken om deze handeling over een aantal opdrachten te verbieden -en dit scheidt zijn problemen. Dit laatste vloeit voort uit het feit, dat de transputmachine bij autonoom geheugencontact absolute prioriteit boven de machine heeft (in overeenstemming met het feit, dat je op dit microscopische niveau essentiële haastsituaties moet kunnen toestaan).

b) dat bij beïnvloeding van  $t$  en  $T$  door de computer het pertinent verboden is als, hoe kort maar ook, ten onrechte  $T = 1$  zou gelden. Op dat moment zou nl. meteen het transput-apparaat ten onrechte gestart kunnen worden.

Dit betekent, dat de computer de  $V$ -operatie niet mag uitvoeren door het programma

```
"t:= t + 1; C:= 0 < t;
  if C then T:= 1"
```

(Opm. De algol-programma'tjes, waarmee we de actie van de machine beschrijven hebben als neven-eigenschap, dat met elke regel 1 X8-opdracht overeenkomt; dit maakt het duidelijker om te zien, waar het transputapparaat -tussen de opdrachten nl.- er tussen door kan komen. Men lette er op, dat de telling aan  $t$  met een (conditiezettende) additieve uitopdracht wordt uitgevoerd; daarmee is althans de telling een ondeelbare handeling en kunnen de tellende operatie's van computer en transputmachine niet met elkaar interfereren.)

Als het bovenstaande programma wordt uitgevoerd op een moment, dat  $T = 1$  is, bestaat er immers de kans, dat na de test op het teken van de nieuwe waarde van  $t$  en voor de assignment  $T := 1$  het transput-apparaat zoveel  $P$ -operaties uitvoert, dat  $t$  tot nul wordt afgelaagd en in overeenstemming daarmee  $T := 0$  verricht wordt, waarna de computer op grond van een inmiddels obsoleet gegeven alsnog  $T := 1$  uitvoert.

De weg uit deze ellende is

```
"t:= t + 1;
  if T = 0 then
    begin if 0 < t
      then T:= 1 end"
```

De rechtvaardiging van dit stukje programma is nu als volgt. Eerst testen we  $T$ ; als  $T = 1$  is, dan was dus  $0 < t$ ; door de optelling is  $t$  alleen maar groter geworden en het eventueel aanpassen van  $T$  ligt nu op de weg van de transputmachine. Maar als we  $T = 0$  aantreffen, dan kan de transputmachine de operatie  $P$  niet meer uitvoeren, zodat van die kant  $t$  en  $T$  ongemoeid gelaten worden en de computer rustig de test kan uitvoeren.

In de praktijk van de X8 zal het nog een stapje ingewikkelder uitgevoerd worden. Een dergelijke "Actie-flio-flop" als  $T$  zal zich, om de communicatielijn niet nodeloos te belasten niet in de basismachine, maar in de transputmachine in kwestie bevinden. Dit maakt de vraagopdracht " $T = 0$ " een minder aantrekkelijke propositie. Daarom wordt in het geheugen een copie van  $T$  bijgehouden,  $ST$  genaamd.

De  $P$ -handeling door het transputapparaat wordt nu:

```
"if T = 1 then begin t:= t - 1; if t < 0 then ST:= T:= 0 end"
```

en de  $V$ -handeling door de computer wordt

```
"t:= t + 1;
  if ST = 0 then
    begin if 0 < t then
      begin ST:= 1;
        T:= 1
      end
    end
  end"
```

waarbij we er op moeten letten, dat de assignment aan ST voor de assignment aan T wordt uitgevoerd.

Om het aantal geheugencontacten van de transputmachine te beperken is het gewenst om ST en t in hetzelfde woord te representeren, maar dan zo, dat de operatie  $t := t + 1$  ST niet wijzigt. Dit kan bv. door het woord in te delen:

$$d[26] := ST$$

$$d[25] \dots d[0] := (2^{25} - 1) + t$$

De test " $t \leq 0$ " komt dan neer op de test " $d[25] = 0$ ".

## 2.2. De V-operatie door de transputmachine.

Weer zullen we de seinpaal aanduiden met t en de bijbehorende flip-flop met T en zal gelden  $T = 1$  als  $0 < t$ . De V-operatie is nu eenvoudig

" $t := t + 1$ ; if  $0 < t$  then  $T := 1$ "

door het transput-mechanisme uit te voeren als een ondeelbare, door de computer niet tegen te houden handeling.

De P-operatie zal de computer alleen uitvoeren als  $T = 1$ ; de computer kan zich, bij gratie van doofheid, echter permitteren, om tijdelijk  $T = 0$  te zetten.

De voltooiing van de P-operatie door de computer bestaat nu uit het volgende stukje programma

```
"t := t - 1;
  T := 0;
  if  $0 < t$  then
    T := 1"
```

Essentieel is hierbij aangenomen

- dat het geen kwaad kan, als T ten onrechte een tijdje = 0 is;
- dat als transput-machine en computer "tegelijkertijd" proberen om de assignment " $T := 1$ " uit te voeren, dat dan na afloop zeker " $T = 1$ " zal gelden.

Om aan te tonen, dat bovenstaand stukje programma na afloop onder alle omstandigheden de waarde T in overeenstemming met t achterlaat, hoeven we niet te veronderstellen, dat T aanvankelijk positief was. (Als we de eerste twee statements omdraaien, dan moet ik dat wel aannemen; een helderder inzicht in dit soort spelletjes is nog steeds zeer gewenst!) We kunnen desgewenst dit programma ook gebruiken om via " $t := t - n$ " een zg. negatieve voorgift te geven.

De redenering is als volgt. Elke V-operatie voor de assignment  $T := 0$  uit bovenstaande programma'tje doet niet ter zake, omdat daarna in elk geval  $T := 0$  uitgevoerd wordt. Als tijdens de assignment  $T := 0$  t positief is, dan zal de computer uiteindelijk de statement  $T := 1$  uitvoeren. Door tussengeschoven operatie's V kan t immers dan alleen maar nog groter worden en T wordt dus correct achtergelaten. Als tijdens de assignment  $T := 0$  daarentegen t non-positief is, dan moeten wij twee gevallen onderscheiden. Als ~~XXXXX~~ ondanks eventuele tussengeschoven V-operaties t non-positief blijft, dan zullen zowel transputmachine als computer T verder ongemoeid laten en dus zal T dus correct = 0 achtergelaten worden. Als door ondergeschoven V-operaties daarentegen t positief wordt, dan zal in elk geval door de transputmachine op het moment van omslag  $T := 1$  uitgevoerd worden (en mogelijk ten overvloede ook nog een keer door de computer, nl. als t ten tijde van de test al positief is). In dit laatste geval wordt T dus gegarandeerd correct = 1 achtergelaten.

Na deze analyses van V- resp. P-operatie door de computer vertrouw ik

- a) dat de lezer zich van de waterdichtheid overtuigd heeft
- b) dat de lezer mij de uitvoerigheid, waarmee ik dit heb uitgesponnen, niet kwalijk neemt. Integendeel zelfs.

Als de P-operatie door de machine uitgevoerd moet worden, dan is de bijbehorende flip-flop T een zg. ingreep-flip-flop. Een ingreep-flip-flop zit in de basismachine, kan door de computer in beide richtingen gezet worden, kan door de transputmachine op 1 gezet worden. (Dit in tegenstelling tot een actie-flip-flop. zie 2.1.)

### 2.3. Hardware ten behoeve van de ingreep-controle.

Ingreep-flip-flops kunnen (zie 2.2.) individueel in beide richtingen gezet worden.

Aan elke ingreep-flip-flop is een luisterbit toegevoegd. Ook deze is als flip-flop uitgevoerd; ook de luisterbits kunnen individueel in beide richtingen gezet worden door de computer.

Ingreep-flip-flops, zowel als luisterbits kunnen woordsgeijs worden uitgelezen via een communicatie-opdracht. (Zolang het er minder dan 27 per type zijn, dan doen we dit met een woord voor elk type) Overeenkomstige flip-flops hebben in deze woorden dezelfde positie.

Ten slotte is er een doof-horendbit in de computer, die tevens beschikt over de opdrachten "maak doof" en "maak horend".

Als voldaan is aan de volgende twee voorwaarden:

- a) de machine is horend
- b) het collatie-resultaat van ingreepwoord(en) en (overeenkomstige) luisterwoord(en) is  $\neq 0$

dan vindt een ingreep plaats, dwz.

inplaats van normaal de volgende opdracht aan te halen, wordt een subroutine-sprong in het opdrachtregister geplaatst, die met onderdrukking van de ophoging van de opdrachtsteller wordt uitgevoerd; tevens wordt de machine door deze actie automatisch doof.

Opm. Hieruit zou volgen, dat een ingreep alleen maar plaats kan vinden op een plek in het programma, waar de machine horend is. Als tijdens de laatste opdracht voor de ingreep al besloten is, dat deze ingreep plaats zal vinden, dan is het mogelijk moeilijk om op deze decisie terug te komen, als de laatste opdracht nu juist de opdracht "maak doof" was. De oplossing is om bij het wegbergen van de link ook de "doof-horendheid" te redden en bij de voortzetting van het onderbroken programma "herstellend" terug te springen. Bij de X1 is gebleken, dat "vertraagde actie" van de opdrachten "maak doof" en "maak horend" mits deze vertraging zich slechts over 1 opdracht uitstrekt, geen enkele moeilijkheid oplevert.

Het programma'tje voor de aflaying van T in 2.2. zal in doofheid geschieden; vandaar dat T vrijelijk even een verkeerde waarde mag hebben.

De luisterbits stellen ons in staat ingreep-flip-flops -eventueel tijdelijk- te negeren, wanneer er nl. -nog- geen belangstelling voor bestaat.