Werkexemplaar

REFERENCES versus POINTERS

ofwel: wees wijzer met wijzers en verwijzingen

Maarten Fokkinga, 26 maart 1984

Abstract "Mijn Algol 68 opvatting" en "de Pascal opvatting" van het begrip assigneerbare variabele worden beschreven en met voorbeelden en plaatjes toegelicht. Het komt erop neer dat in mijn Algol 68 opvatting cellen van de opslagruimte met hun adressen worden geidentificeerd, terwijl ze in Pascal worden onderscheiden als verschillende datatypen.

Met dank aan Joost Engelfriet die commentaar gaf op een eerder verhaal en me daardoor tot deze uiteenzetting inspireerde. En ook in deze versie foutjes aanwees.

* * *

In wat hieronder volgt betreft de linkerkolom steeds Algol 68 en de rechter Pascal.

ALGOL 68	PASCAL

We gaan uit van de opvatting dat de opslagruimte van een machine is verdeeld in cellen, en dat bij iedere cel een uniek adres hoort.

In Algol 68 worden cellen met hun adressen geidentificeerd: zo'n adres, dat dus tevens dienst doet als cel, noem ik een reference ~~wordt genoemd~~.

In Pascal worden cellen en adressen onderscheiden. Een cel wordt variabele genoemd, zijn adres pointer.

We laten in het vervolg r varieren over references, v over variabelen, en a over pointers. We gebruiken in de syntaxis x, xx, y, yy, p en q als identifiers die references, variabelen of pointers aanduiden.

In onderstaande tabel noteren we symbolisch de semantische bewerkingen, testen en constante van de datatypen. Een toelichting volgt na de tabel.

voor REFERENCES r, r'	voor VARIABELEN v, v'
R1. create (levert een nieuwe r)	V1. create (levert een nieuwe v)
R2. r := ...	V2. v := ...
R3. deref(r)	V3. deref(v)
R4. r = r'	V4. pointer_to(v) (levert een a)
R5. ... := r (r is storable)	
R6. nil (een speciale r)	

voor POINTERS a, a'	
P1.	a = a'
P2.	a↑ (levert een v)
P3.	... := a (a is storable)
P4.	nil (een speciale a)

Toelichting.

R1, V1: We zien af van het aspect van de levensduur ofwel extent. Dus deze bewerking delt de creatie die door loc, heap, var-declaratie, en binnen new( ) plaats vindt.

R2, V2: In overeenstemming met het informele machine model is de bestemming van een assignement in Pascal een cel (= variabele), en geen adres (= pointer). Vanwege de identificatie tussen cellen en adressen is de bestemming in Algol 68 een reference.

R3, V3: Dereferencing is: het uitlezen van de inhoud van een cel. Het woord "dereferencen" komt uit het Algol 68 jargon. In het Pascal rapport wordt er geen naam voor gegeven; wel wordt gesproken over "the current value of a variable". Kortheidshalve heb ik die operatie op variabelen ook maar met deref genoteerd. Voor beide talen geldt dat dereferencing niet expliciet in de programmatekst geschreven kan worden.
Wij zullen in het vervolg alle dereferencing operaties expliciet aangeven.

V4: De bewerking pointer_to kan in Pascal niet vrijelijk door de programmeur gebruikt worden, maar vindt slechts en alleen impliciet plaats binnen new:

```
procedure new (var p: ↑T);
  begin   create a new v;
          p := pointer_to (v)
  end
```


Vanwege de identificatie van cellen en adressen is de pointer_to operatie in Algol 68 niet nodig; desgewenst kun je hem simuleren met de identiteitsfunktie!

R5, P3: Intuitief valt er wat voor te zeggen dat cellen niet "storable" zijn, en adressen wel. De Pascal opvatting stemt hiermee overeen. Ten gevolge van de identificatie van cellen met adressen kan in Algol 68 deze onderscheiding niet aangebracht worden.

R4, P1: Let erop dat de gelijkheidstest tussen references syntactisch niet met = maar met is wordt genoteerd. In Pascal is er geen gelijkheidstest op cellen; dus aliassing (twee namen voor eenzelfde variabele) kan in Pascal niet getest worden ~~(in het algemeen)~~! (behalve ~~als je~~ door de pointer_to die celle ~~op mogelijkheid te toets tot je bevalling luidt~~)

P2: Pascal suggereert geen andere terminologie voor a↑ dan: the variable referenced by a. Ik spreek a↑ uit als: a gevolgd. Let erop dat dit een andere operatie is dan deref uit V3! Vanwege de identificatie tussen cellen en adressen is deze operatie in Algol 68 niet nodig, het zou de identiteit zijn. Voorts geldt dat pointer_to en ↑ elkaars inversen zijn.

R6, P4: nil is een voorgedefinieerde r of a waarop operaties R2, R3 resp P2 niet toegepast mogen worden; voor het overige onderscheidt nil zich niet van andere r of a.

We geven nu enige voorbeelden. De asserties, ofwel beweringen over de toestand, geven we grafisch

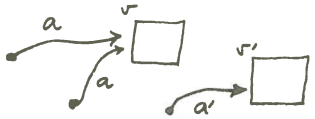
weer. Het zijn bij uitstek de gelijkheid van variabelen, adressen en references, en de ~~relatie~~ deref-funktie die zich grafisch makkelijker **) laten weergeven dan in formules; andere beweringen over de toestand kunnen vaak beter in formules worden gegeven.

Een reference r tekenen we als een lijn die eindigt bij een hokje dat uniek is voor die reference. Dus bij verscheidene tekeningen van r zijn er verscheidene (beginpunten van) lijnen, maar slechts één hokje.

(We geven later nog twee varianten hierop.)

Een variabele tekenen we als een hokje; verschillende hokjes voor verschillende variabelen.

Een pointer tekenen we als een pijl; de punt van de pijl vlak bij de variabele waarvan hij het adres is.



deref(xx) wijst naar positie van het hokje toe

De hokjes stellen de cellen voor, de erbij eindigende lijnen hun adressen. In Algol 68 worden die geïdentificeerd: een reference is een lijn **mét** het hokje. De inhoud van een cel tekenen we in het hokje; die inhoud wordt door deref opgeleverd.

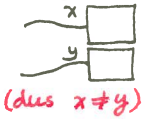
**) zie PAG 11.

✓ Preciezer: het beginpunt van zo'n lijn is "de tekening" van de reference.

✓ Preciezer: het beginpunt van zo'n pijl is "de tekening" van de pointer.

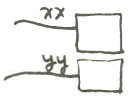
In de voorbeelden hieronder zetten we soms een of meer programma-expressies bij de daardoor aangeduide objecten. Afwisselend volgen nu programmateksten en toestandsbeweringen.

ref int x, y

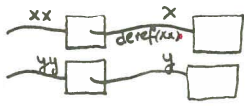


(dus $x \neq y$)

ref ref int xx, yy

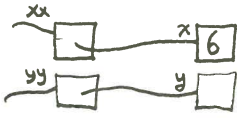


xx := x; yy := y



deref (xx) := 6

(xx éénmaal dereferencen)

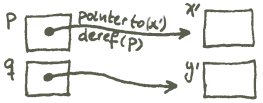


var x, y: int

(dus x en y ~~zijn~~ verschillen)

var p, q : ↑ int

new(p) i.e. var x' :int; p := pointer.to(x')
new(q)



deref(p) ↑ := 6

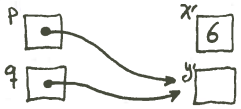
(p eerst dereferencen, dan de pointer volgen!)

xx := deref(yy)

dus nu

$x \neq deref(xx) = deref(yy) = y$

p := deref(q)

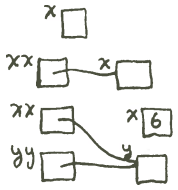


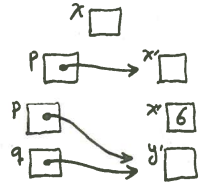
dus nu

pointer_to(x') ≠ deref(p) = deref(q) = pointer_to(y')

én: deref(p)↑, deref(q)↑, y' zijn alias~~derelfde~~

Mijn Algol 68 plaatjes zijn anders dan (maar formeel wel isomorf met) die van Van der Meulen en Kühling. Het voordeel van mijn plaatjes is dat ze beter aansluiten bij die van Pascal, als je de volgende conventie navolgt: Wanneer het "beginpunt van een reference" niet terzake doet (zoals hierboven bij alle tekeningen van xx en yy, en de eerste en laatste tekening van x), tekenen we die lijn naar het hokje héééééél kort, zó kort dat je hem niet meer ziet. We krijgen dan:





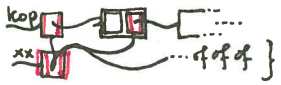
Desgewenst kunnen we de ref-nivo's ook nog in

het plaatje weergeven, bijvoorbeeld

In Pascal nauwelijks nodig, omdat een ↑↑T type nauwelijks voorkomt.

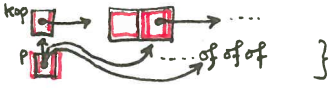
De reden dat een ↑↑T type nauwelijks in Pascal voorkomt, is dat de operatie pointer_to uitsluitend impliciet binnen new wordt toegepast; hij is met name niet van toepassing op anderszins gecreëerde variabelen of componenten van variabelen. Dit is een paternalistische heuke van de taalontwerper geweest; er is semantisch gezien geen reden om het gebruik van pointer_to zo te beperken. Er bestaat natuurlijk wel het gevaar van zg. dangling pointers bij een vrij gebruik van pointer_to, maar daartegenover staat weer het gemak bij het programmeren van bijv. lijstverwerkingen. Bijvoorbeeld, om een lijst achteraan met een gegeven element uit te breiden:

```
mode m = ref struct(..., m next);
m elem = ....
ref m hop = ....
ref ref m xx = loc ref m := hop;
while deref(deref(xx)) ≠ nil
do { hop ... of of of }
```



zie pag 13

```
type t = ↑ record ... ; next: ↑t end;
elem : t ....
var hop: t ....
var p: ↑t := pointer_to (hop);
while deref (deref(p)↑) ≠ nil
do { hop ... of of of }
```



```
xx := next of deref(deref(xx))          p := (deref(deref(p)↑)↑. next)
od;                                     od;
deref(xx) := elem                       deref(p)↑ := elem
```

Bedenk dat in beide talen de deref-bewerkingen meestal niet aangegeven hoeven worden. Dus feitelijk worden de toekenningen binnen do-od geschreven als

```
xx := next of xx                        p := (p↑↑. next)
```

De deref in deref(xx):=... moet in Algol 68 door een cast worden afgedwongen; in Pascal hoeft dat niet:

```
ref m (xx) := elem                      p↑ := elem
```

Zonder het vrije gebruik van pointer_to moet het geval van een lege lijst afzonderlijk behandeld worden, en is de waarde van p geen pointer naar een next-veld, maar naar de variabele die dat next-veld omvat. Dat geeft wat meer schrijfwerk en is ook ietwat minder efficient (er vinden meer veld-selecties plaats). En ook al mag een programmeur de bewerking pointer_to niet expliciet gebruiken, hij moet die operatie wel kennen om de semantiek-definitie (of: -suggesties) van het Pascal rapport te

begrijpen! (Zie bijvoorbeeld de "definitie" van new(p).)

Tot besluit

Soms vindt men wel eens dat Pascal eenvoudiger is ten aanzien van "pointerarithmetiek" dan Algol 68. Dat komt mijns inziens door de volgende redenen.

1. Semantisch. De onderscheiding tussen cellen en hun adressen is intuitief wel aantrekkelijk. Dus de bestemming van een assignment is een cel (en geen adres), en cellen zelf zijn niet "storable" maar hun adressen wel. Toch is ook de identificatie van cellen met adressen niet zo gek, te meer daar iedere bit-representatie van een cel toch zijn (absoluut of relatief) adres is.
2. Syntactisch. In Pascal wordt van een variabele altijd zijn waarde genomen, behalve wanneer hij optreedt als bestemming van een assignment (of als var-argument *)). In andere woorden, behalve op de bestemmingspositie van een assignment of op een var-positie in een argumentenlijst, wordt deref

*) zie pag 11.

3 Beperkingen op pointer to.
- plaatjes, dangling references vs mogelijken progr's
- alias test niet mog'k
* toch kennen

4 Terminologisch

altijd (precies eenmaal) automatisch tussengevoegd. In Algol 68 gebeurt dit nul, één, of meermalen. En erger nog, het aantal dereferencing coercies wordt in Algol 68 door het type (= de mode) van de hoofdlist bepaald, terwijl dat in Pascal door de vorm van de onmiddellijke hoofdlist wordt bepaald. Wat dit betreft is Pascal dus echt eenvoudiger dan Algol 68.

Daar komt bovendien nog bij dat "de Algol 68 opvatting" zoals die uit de formele terminologie van het Algol 68 rapport blijkt, helemaal tegen-intuitief is. Want in die opvatting is een reference (officieel: een 'name') geen verwijzing naar een cel, maar:

> a value which can be "made to refer to" some other value...

dus: een verwijzing naar een waarde. Bijgevolg verwacht de assignment-operatie een verwijzing naar een waarde als bestemmings-operand. Dit vind ik tegen-intuitief, hoewel 't formeel allemaal wel klopt. Deze opvatting wordt door Van der Meulen uitgedragen in zijn plaatjes, en ook door Koster ~~in zijn leerboeken~~ in zijn leerboeken. Maar gelukkig wordt mijn opvatting ondersteund door het (niet tot de definitie behorende)

kommentaar in het Algol 68 rapport dat een reference

> ...may be thought of as the address of the storage cell in the computer used to contain the value ...

(Revised Report, 2.1.3.2.a).

— * * * —

---

*) De selectie en subscriptie moeten gezien worden als funkties met een var-argument en een var-resultaat (géén pointer resultaat). Dit is zo omdat bijv. selectie op een record-variabele een komponent-variabele oplevert. Ditzelfde geldt ook in Algol 68.

**) Gelijkheid of verschil van variabelen kan niet eens i.h.a. in Pascal-formules uitgedrukt worden, dus "moet" je wel je toevlucht nemen tot plaatjes. Algol 68 is krachtig genoeg om de plaatjes ook in formules te beschrijven.

[toevoeging dd 29 mrt 84][References versus Pointers]

Wat de grotere uitdrukkingskracht van Algol 68 betreft, merken we op dat formeel alleen de alias-test niet in Pascal en wel in Algol 68 geprogrammeerd kan worden. Het maken van aliassen kan in beide talen, zij het dat daarvoor in Pascal het procedure mechanisme nodig is. Praktisch gesproken is Algol 68 daarin beter --en efficiënter-- dan Pascal. We geven hiervan een voorbeeld. Stel dat

```
(next of a[i][i+j])[k]                a[i][i+j].next [k]
```

een m-reference resp. een t-variabele aanduidt. Omwille van de efficiëntie kan het wenselijk zijn zo'n uitdrukking éénmaal uit te rekenen en te benoemen of op te slaan, om hem dan vele malen te gebruiken:

```
(-----
ref m x = (next of a[i][i+j])[k];
---deref(x)---
--- x:=---deref(x)---
)
```

```
(----
 p ( a[i][i+j].next[k])
)
met
proc p (var x: t);
begin ---deref(x)---
      --- x:= ---deref(x)---
end
```

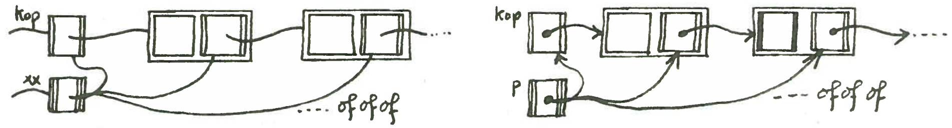

Ware het zo geweest dat pointer-to vrijelijk gebruikt mocht worden in Pascal, dan hadden we het procedure mechanisme niet hoeven gebruiken. (De Algol 68 tekst moeten we ook iets aanpassen om volledige overeenstemming te bereiken, zie de declaratie van x.)

```
(----
ref m x := (next of a[i][i+j])[k];
{deref(x) is (next of a[i][i+j])[k]}
--- deref(deref(x))---
--- deref(x) := --- deref(deref(x))
)
```

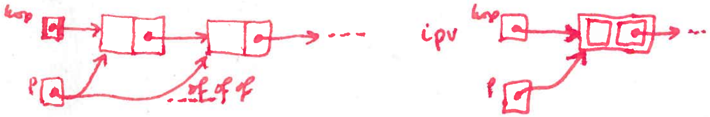
```
(----
var x:t := pointer-to (a[i][i+j].next[k]);
{deref(x)↑ en a[i][i+j].next[k] zijn aliassen}
--- deref(deref(x)↑)---
--- deref(x)↑ := --- deref(deref(x)↑)...
)
```

(De with-statement in Pascal maakt slechts heel bijzondere aliassen, namelijk alleen voor komponent-variabelen van een record-variabele.)

PS De tekening vd invariant onderaan pag 7 kan iets preciezer:



Omdat pointer-to alleen werkt op gehele variabelen, hoeven we in Pascal de deelvariabelen nooit afzonderlijk als hokjes te tekenen. Dus bv.



(Geen misverstand mogelijk: pijl naar 'n deelvariabele uitgesloten)