Werkexemplaar

Assignments in Asserties

toegelicht met de Schorr-Waite algoritme

Maarten M Fokkinga

17 okt 1984

Asserties zijn beweringen over de toestand (op één tijdstip) gedurende de berekening, en worden gebruikt om de korrektheid van algoritmen aan te tonen. Gewoonlijk worden asserties geformuleerd in een wiskundig-achtige taal of een zeer informele versie daarvan; het gebruik van "neveneffecten" binnen asserties is hoogst zeldzaam, maar allerminst onmogelijk of inconsistent. We zullen de Schorr-Waite algoritme korrekt bewijzen door in de asserties assignments, en programmafragmenten op te nemen.

Inleiding

Beschouw eens hetvolgende eenvoudige probleem: een gegeven rij a[1:n] te sommeren. Een geannoteerd programma hiervoor is

$s, i := 0, 0;$

$\underline{\text{do}}\ \{ s = a[1]+\cdots+a[i] \}\ i \neq n \rightarrow i, s := i+1, s+a[i+1]\ \underline{\text{od}}$

De assertie, in dit geval zelfs invariant van de repetitie is: $s = a[1]+\cdots+a[i]$. Dit is een informele uitdrukking in de taal der wiskunde. Een alternatieve formulering is

$$\text{gevraagde som} = s + (a[i+1]+\cdots+a[n])$$

Alweer een uitdrukking in de taal der wiskunde, maar nog iets informeler dan de vorige assertie: 'gevraagde som' is impliciet bekend verondersteld; (We hadden het kunnen definieren als $a[1]+\cdots+a[n]$). Nog een andere invariant luidt:

"om de gewenste som te bereiken hoef je nog alleen maar de elementen van $a[i+1:n]$ bij s op te tellen"

Ook hiermee is het programma korrekt te bewijzen. Deze laatste invariant kunnen we ook formuleren als

$\underline{\text{for}}\ j\ \underline{\text{from}}\ i+1\ \underline{\text{to}}\ n\ \underline{\text{do}}\ s \mathrel{+}:= a[j]\ \underline{\text{od}}$
levert na afloop de gevraagde som in s.

We zien nu dat er assignments (en nog meer programmaconstructies) tevoorschijn komen in asserties. Korrektheidsbewijzen blijven hiermee mogelijk! (In Dynamic Logic wordt dit alles zeer formeel bestudeerd) Het is bijvoorbeeld direkt duidelijk dat uit deze laatste invariant, tesamen met $i=n$, volgt dat het programma zijn taak heeft vervuld: de gevraagde som zit in s opgeslagen. Ook de invariantie onder de statement

$$i, s := i+1, \; s + a[i+1]$$

is bijna triviaal. En initieel is bij $i=0$, $s=0$ de invariant óók waar; alleen is die waarheid even moeilijk formeel te bewijzen als de korrektheid van het programma zelf. (Tenzij de programmaspecificatie, de postassertie dus, al geformuleerd was met gebruikmaking van zo'n for-statement!!)

Heel algemeen gesteld kunnen we als invariant voor het programma

$$\{P\} \; \underline{\text{while}} \; B \; \underline{\text{do}} \; S \; \underline{\text{od}} \; \{Q\}$$

nemen:

"verdere executie van dit programma zelf zal tot Q leiden (indien terminerend)"

ofwel: "de toestand is zodanig dat $\underline{\text{while}}$ B $\underline{\text{do}}$ S $\underline{\text{od}}$ $\{Q\}$"

Uit deze invariant volgt, tesamen met $\neg B$, onmiddellijk Q en ook de invariantie is triviaal. De volledige bewijslast komt in feite neer op het bewijzen dat initieel P de invariant waar maakt. Met zo'n invariant schiet je dus niet veel op.

We kunnen ook als invariant nemen:

"deze toestand is ontstaan uit P door nul of meer slagen van de herhaling"

Nu is de initialisatie triviaal, evenals wederom de invariantie. De hele bewijslast komt nu neer op het aantonen dat uit die invariant, tesamen met $\neg B$, de postassertie Q geldt. Ook hier schiet je dus niet veel op.

We zullen straks, bij de Schorr-Waite algoritme, een invariant geven waarvan wél de initialisatie, finalisatie en invariantie niet-triviaal zijn. De reden dat we onze toevlucht nemen tot "imperatieve asserties" is dat er met pointer-structuren gewerkt wordt. Het blijkt vooralsnog gemakkelijker om uit te drukken welke acties nog ondernomen moeten worden, dan om de toestand te relateren aan bijvoorbeeld de eindtoestand. Dit laatste houdt namelijk in dat niet alleen de inhouden der records beschreven wordt, maar ook de posities van de records in de opslagruimte (nl. de pointerwaarden zelf!). En het is juist in dit opzicht dat de non-imperatieve assertietaal te kort schiet. Misschien dat er nog goede notaties bedacht worden om pointerstructuren te beschrijven; vooralsnog zijn die er niet.

De Schorr-Waite algoritme

De Schorr-Waite algoritme is een iteratief algoritme dat zonder extra opslagruimte een binaire boom doorloopt en in iedere knoop een markeringsbit, initieel 0 voor alle knopen in de opslagruimte, op 1 zet. ~~Het~~ De algoritme is dus uitermate geschikt voor de garbage dectection-fase in een garbage collector.

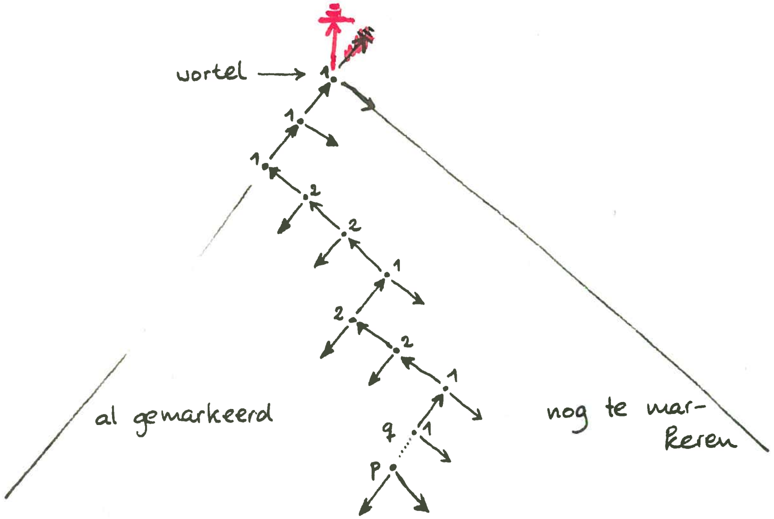
We zullen geen poging doen de algoritme af te leiden uit de recursieve boomwandeling; we volstaan met de presentatie van de uiteindelijke vorm van de algoritme en z'n informele --maar hopelijk wel overtuigende-- korrektheidsbewijs.

We gaan uit van de volgende definities

```
type knoop = record v: integer;
                    l, r: ↑knoop;
                    --- overige informatie
             end;
var p, q, k : ↑knoop
```

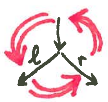
Het v-veld van een knoop is de markeringsbit. Wij zullen hieronder v de waarden 0, 1, 2, 3 geven, maar in feite mag ~~iedere~~ die waarden modulo 2 genomen worden, zodat v inderdaad slechts de ruimte van één bit inneemt. Wanneer, in onze formulering van de algoritme, i de waarde van v is, dan is de betreffende knoop precies i keer bezocht (v = visit). Dus, met dat ene extra v-veld (met waarden 0..3) kan een pre/in/post-order boomwandeling geprogrammeerd worden zonder extra stapel! (Overigens, als er per knoop niet twee subbomen zijn, genaamd l en r, maar een heel stel, zeg sub[1..n], dan kan de algoritme daaraan aangepast worden; v moet dan de waarden 0..n+1 kunnen aannemen. Ook de generalisatie tot circulaire structuren is mogelijk, mits daarvoor per knoop een veld extra beschikbaar is waarin opgeslagen kan worden of de knoop al eens is binnengegaan --vanuit een "vader", niet: vanuit een "zoon"--.)

Natuurlijk is er wel een stapel ergens gecodeerd aanwezig: we (mis)bruiken daartoe de l- en r-velden van de knopen op het pad vanuit de huidige knoop q tot aan de wortel:



Het getal bij ~~iedere~~ een knoop geeft de waarde van v aan. Als $p\uparrow.v = 0$ moet ook de boom p nog gemarkeerd worden; als $p\uparrow.v = 3$ is p geheel gemarkeerd.

We maken de volgende, vrij willekeurige, representatiekeuze. Zij k een knoop op het pad van q tot de wortel, dan hebben de drie pijlen $k\uparrow.l$, $k\uparrow.r$ en "pijl naar k toe" de gebruikelijke orientatie:



en bv.

Het plannetje laat zich nu als volgt in woorden formuleren; dit geeft in feite het belangrijkste bestandsdeel van de invariant.

(1) definieer pad(q) als volgt:

$$\text{pad}(q) = (k_0, k_1, \ldots k_n) \quad \text{met}$$

$$k_0 = q$$

$$k_{i+1} = \begin{cases} k_i\uparrow.r & \text{als } k_i\uparrow.v = 1 \\ k_i\uparrow.l & \text{als } k_i\uparrow.v = 2 \end{cases} \qquad \text{voor } k_i \neq \underline{\text{nil}}$$

(2) definieer voor iedere knoop $k \in \text{pad}(q)$, $k \neq \text{nil}$, vader (k) als volgt:

$$\text{vader}(k_0) = p$$

$$\text{vader}(k_{i+1}) = k_i$$

P0:

> De gewenste eindboom wordt verkregen uit het huidige ~~drie~~2tal (~~wortel,~~ p, q) --en ~~in feite~~ de hele opslagruimte-- door
>
> (i) van iedere $k \in \text{pad}(q)$ met $k \neq \underline{\text{nil}}$
>
> - als $k\uparrow.v = 1$ nog $k\uparrow$ zelf en geheel boom $k\uparrow.l$ te markeren,
> - als $k\uparrow.v = 2$ nog $k\uparrow$ zelf te markeren; en
>
> (ii) voor iedere $k \in \text{pad}(q)$ met $k \neq \text{nil}$ nog te doen:
>
> $\underline{\text{if}}\ k\uparrow.v = 1 \rightarrow k\uparrow.l, k\uparrow.r := \text{vader}(k), k\uparrow.l$
>
> $[]\ k\uparrow.v = 2 \rightarrow k\uparrow.l, k\uparrow.r := k\uparrow.r, \text{vader}(k)$
>
> $\underline{\text{fi}}$
>
> (iii) geheel boom p te markeren indien $p \neq \underline{\text{nil}} \wedge p\uparrow.v = 0$

P0:

gew. eindsituatie treedt op na afloop van:

~~[illegible]~~ $\underline{\text{if}}\ p\uparrow.v = 0 \rightarrow \text{mark}(p)\ []\ p\uparrow.v = 3 \rightarrow \text{skip}\ \underline{\text{fi}};$

$\underline{\text{do}}\ q \neq \underline{\text{nil}} \rightarrow \underline{\text{if}}\ q\uparrow.v = 1 \rightarrow \text{mark}(q\uparrow.l)$ ~~[illegible]~~;

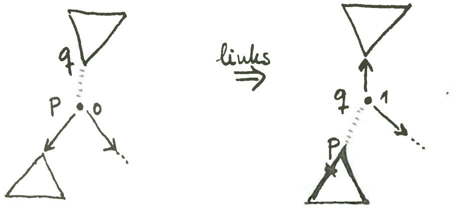
$(p, q, q\uparrow.l, q\uparrow.r, q\uparrow.v) := (q, q\uparrow.r, p, q\uparrow.l, 3)$

$[]\ q\uparrow.v = 2 \rightarrow$ ~~[illegible]~~ $(p, q, q\uparrow.l, q\uparrow.r, q\uparrow.v) := (q, q\uparrow.l, q\uparrow.r, p, 3)$

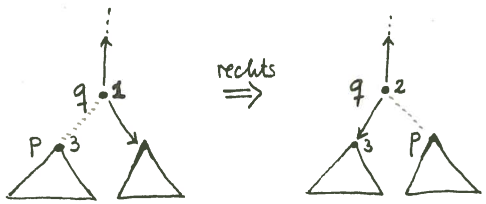
$\underline{\text{fi}}$

$\underline{\text{od}}$

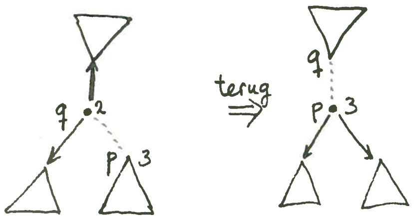
De algoritme bestaat uit het herhaaldelijk verplaatsen van (p, q) over de boom, volgens de normale boomwandelingvolgorde. Er zijn drie verplaatsingsmogelijkheden: links naar beneden, rechts naar beneden en terug omhoog. In figuur en in programmatekst luiden zij als volgt.



$p \neq nil \wedge p\uparrow.v = 0$
$\rightarrow$
$(p,\ p\uparrow.l,\ p\uparrow.r,\ p\uparrow.v,\ q) := (p.l,\ p\uparrow.r,\ q,\ 1,\ p)$



$q \neq nil \wedge q\uparrow.v = 1 \wedge (p = nil \vee p\uparrow.v = 3)$
$\rightarrow$
$(p,\ q\uparrow.l,\ q\uparrow.r,\ q\uparrow.v) := (q\uparrow.l,\ q\uparrow.r,\ p,\ 2)$



$q \neq nil \wedge q\uparrow.v = 2 \wedge (p = nil \vee p\uparrow.v = 3)$
$\rightarrow$
$(p,\ q\uparrow.l,\ q\uparrow.r,\ q\uparrow.v) := (q,\ q\uparrow.r,\ p,\ 3)$

(Een als $\triangle$ of ∇ getekende boom (pointer) mag ook nil zijn; in dat geval is een eventueel aangegeven v-waarde niet van toepassing.)

Bovenstaande guarded commands zijn korrekt: de invariant P_0 wordt niet verstoord door de actie indien de guard slaagt. De imperatieve formulering van assertie P_0 laat dit gemakkelijk verifiëren. Bovendien is de volgende assertie óók invariant (en in feite impliciet in P_0 !):

P_1: voor alle knopen $k \in pad(q)$ met $k \neq \underline{nil}$
geldt $k\uparrow.v = 1 \vee k\uparrow.v = 2$, en
voor p geldt: $p = \underline{nil} \vee p\uparrow.v = 0 \vee p\uparrow.v = 3$

De verificatie hiervan is ook bijna triviaal. (Bedenk dat alle v-waarden initieel 0 zijn.)

Het programma

$p, q :=$ wortel, $\underline{nil}$;
$\underline{do}$ links ▯ rechts ▯ terug $\underline{od}$

is korrekt: de initialisatie vestigt $P_0 \wedge P_1$ en uit de terminatie-conditie volgt (m.b.v. P_1) dat $q = nil \wedge (p = nil \vee p\uparrow.v = 3)$, dus volgens P_0 is de gewenste gemarkeerde boom verkregen.

De algoritme kan nog iets versneld worden door de nondeterministische keuze tussen de

guarded commands te beperken, in ruil voor het invariant houden van extra betrekkingen. Hetvolgende programma geeft dit weer.
(links', rechts' en terug' zijn de assignments, zonder guards!)

```
p, q := wortel, nil;
do① p ≠ nil → links' od; ②
do③ q ≠ nil →
    if q↑.v = 1 →④ rechts'; do⑤ p ≠ nil → links' od ⑥
    ▯  q↑.v = 2 →⑦ terug' ⑧
    fi
od
```

Behalve de invariantie van $P_0 \wedge P_1$ zijn er ook de volgende asserties/invarianten:

①: $p = nil \vee p\uparrow.v = 0$, dus t.z.t. guard van links' vervuld
②: $p = nil$, dus ③
③: $p = nil \vee p\uparrow.v = 3$
④: guard van rechts: $q \neq nil \wedge q\uparrow.v = 1 \wedge (p = nil \vee p\uparrow.v = 3)$
⑤: $p = nil \vee p\uparrow.v = 0$, dus t.z.t. guard van links' vervuld
⑥: $p = nil$, dus ③ weer hersteld
⑦: guard van terug': $q \neq nil \wedge q\uparrow.v = 2 \wedge p = nil \vee p\uparrow.v = 3$
⑧: $p\uparrow.v = 3$, dus ③ weer hersteld

NB. Formele verificatie noodzaakt tot het uitbreiden van de invariant $P_0 \wedge P_1$ met P_2 die zegt dat voor

willekeurige knoop k in de boom geldt: als $k\uparrow.v = 3$ dan hebben al zijn zonen ook een v-waarde 3 ("gemarkeerd"), als $k\uparrow.v = 0$ dan zo ook voor al zijn zonen ("nog ongemarkeerd"), en als $k\uparrow.v = 2$ dan is $k\uparrow.r$ geheel gemarkeerd, en als $k\uparrow.v = 1$ dan is $k\uparrow.l$ nog geheel ongemarkeerd. (Dit ligt eigenlijk al besloten in P_0).

Er rest nog slechts op te merken dat we de waarden van de v-velden ook modulo 2 kunnen nemen: dit kan zonder bezwaar in de programmateksten en in de asserties gebeuren, zonder tegenspraken. Het enige verschil is dat uiteindelijk niet alle v-velden op 3 gezet zijn, maar slechts op 1. Voor een markerings algoritme is dit precies goed.

Tot besluit.

Alle informatie over de Schorr-Waite algoritme heb ik uit (deRoever 1975). ~~Pas~~ Ondanks dat die een formeel korrektheidsbewijs levert, ben ik de algoritme pas goed gaan begrijpen na mijn imperatieve formulering van de invariant, en mijn presentatie van de algoritme. Ik wens de lezer toe dat die niet óók een eigen verhaal moet schrijven om de algoritme te begrijpen.

Literatuur

de Roever, W.P.: Een correctheidsbewijs van de Schorr-Waite markeringsalgoritme voor binaire bomen. Math. Centre Syllabus 21 (1975) V, pp 79-92

Schorr, H. & Waite, W.M.: An efficient machine-independent procedure for garbage collection in various list structures. Comm ACM 10 (1967) pp 501-506

Harel, D.: First order Dynamic Logic. Springer-Verlag (Berlin etc), LNCS 68 (1979) 133 pages.