Korrekthed af Algoritmer

med fokus på while-løkker

Kim Skak Larsen

Institut for Matematik og Datalogi
Syddansk Universitet, Odense

September 2001

Introduktion

Denne note supplerer korrekthedsafsnittet i [1], som er den anvendte lærebog i Algoritmer og Datastrukturer (DM02), efteråret 2001.

Korrekthed

Vi fokuserer på korrekthed af algoritmer og programmer, der forventes at foretage en beregning og derefter standse. Det overhovedet at standse bliver en del af korrekthedskriteriet. Traditionelt bruges ordet *terminering* om det at standse.

For at adskille termineringsproblematikken fra resten af korrekthedsargumentationen indføres begrebet *partiel korrekthed*, som defineres til at betyde, at en given algoritme er korrekt (giver det korrekte resultat), *hvis* den nogen sinde terminerer.

Dermed kan korrekthed defineres som følger:

En algoritme er *korrekt*, hvis

1. algoritmen terminerer, og
2. algoritmen er partielt korrekt.

Partiel korrekthed

Som i [1] baseres partiel korrekthed på præ- og postbetingelser som omslutter en algoritme, hvor algoritme her refererer til en vilkårlig beregning. Dvs. at en algoritme kan være en kompliceret størrelse eller den kan være eksempelvis en enkelt if-sætning eller en tildeling af en værdi til en variabel.

En algoritme er *partielt korrekt*, hvis der for enhver udførelse, for hvilken præbetingelsen er opfyldt, gælder, at hvis postbetingelsen nås, så er den også opfyldt.

Hvis vi lader `Alg`, Pre og $Post$ betegne henholdsvis algoritmen, præbetingelsen og postbetingelsen, noterer vi ovenstående som $\boxed{Pre}\,\texttt{Alg}\,\boxed{Post}$.

Der kan i en algoritme være mange præ- og postbetingelser i sving, idet enhver lille del af algoritmen (f.eks. en tildeling) har sine egne krav og resultater. Hvis ikke andet er nævnt, vil vi dog antage, at disse betegnelser refererer til den komplette algoritmes præbetingelse og postbetingelse.

De simple tilfælde samt rekursion er behandlet fint i [1]. Baseret på dette fokuserer vi blot på while-løkker. Problemet med en while-løkke, der er omkranset af et sæt præ- og postbetingelser, er, at det er et alt for voldsomt direkte logisk skridt at argumentere for, at postbetingelsen gælder baseret på præbetingelsen, når der er et a priori ubegrænset antal iterationer gennem kroppen af en while-løkke i mellem de to. Af den grund indføres en ny type udsagn, kaldet *invarianter*, hvis opgave det er at udtrykke sammenhænge mellem de variabler, der ændrer sig dynamisk i løkken. Invarianten knytter sig ligesom præ- og postbetingelser til en bestemt position i algoritmen; nemlig positionen umiddelbart inden det boolske udsagn evalueres.

For at vi kan bruge en invariant til noget, skal den naturligvis være opfyldt (sand),

hver gang vi kommer til den givne position i algoritmen. Der ud over skal den være relevant (stærk nok) til det, vi ønsker at bruge den til; nemlig at vise postbetingelsen.

Nedenfor ses en lille algoritme (faktisk er det en del af et Java-program).

```
// Precondition: p ∈ ℕ
r = 1; q = p;
while /*I*/ (q != 0) // Invariant: r · x^q = x^p
{
  r = r * x;
  q = q - 1;
}
// Postcondition: r = x^p
```

"Input'et" til algoritmen er x og p, og algoritmen skal beregne x^p; dvs. x opløftet i p'te, hvilket udtrykkes i postbetingelsen. Præbetingelsen er, at p er et ikke-negativt heltal.

Invarianten er et udsagn, der skal gælde hver gang, vi når til positionen markeret med et "I". For nemheds skyld er selve invarianten dog skrevet helt til højre på samme linie.

Inden vi beviser, at ovenstående algoritme er partielt korrekt, vil vi se på en generel skabelon til at bevise dette for while-løkker.

Betragt følgende abstrakte algoritme:

```
// Precondition: Pre
⟨init⟩
while /*I*/ (B) // Invariant: I
{
  ⟨body⟩
}
// Postcondition: Post
```

Her betegner ⟨`init`⟩ alle initialiseringerne, B det boolske udtryk i while-konstruktionen og ⟨`body`⟩ hele while-løkkens krop.

Man kan etablere, at invarianten *gælder*, ved at vise:

1. $\boxed{Pre}\langle\texttt{init}\rangle\boxed{I}$
2. $\boxed{I \wedge B}\langle\texttt{body}\rangle\boxed{I}$

Ovenstående indfanger formelt kravet om, at invarianten skal være opfyldt hver gang, algoritmen kommer til den pågældende position. Første krav udtrykker, at invarianten skal være opfyldt første gang, og andet krav udtaler sig om resten: Hvis vi kommer til invariant-positionen, og det ikke er første gang, må vi have været der før, og på det tidspunkt må B derfor have været opfyldt, ligesom I skal have været opfyldt.

Lad os nu se på, om invarianten for vores eksempel-algoritme gælder. For at holde styr på værdierne af en algoritme-stumps variabler på forskellige tidspunkter bruger vi umærkede variabelnavne til at betegne variablernes værdier *før* udførelsen af en algoritme-stump og mærkede *efter* udførelsen (vi gør det kun for variabler, der faktisk ændrer værdi).

Først skal vi vise:

$$\boxed{p \in \mathbb{N}}\ \texttt{r = 1; q = p;}\ \boxed{r' \cdot x^{q'} = x^p}$$

Ved indsættelse af værdierne $r' = 1$ og $q' = p$ i udtrykket $r' \cdot x^{q'} = x^p$ fås $1 \cdot x^p = x^p$, hvilket er sandt.

Dernæst skal vi vise

$$\boxed{r \cdot x^q = x^p \wedge q \neq 0}\ \texttt{r = r * x; q = q - 1;}\ \boxed{r' \cdot x^{q'} = x^p}$$

Vi antager som allerede forklaret, at første udsagn er sandt og skal så vise det sidste. Fra algoritmen har vi $r' = r \cdot x$ og $q' = q - 1$. Nu kan vi vise

$$r' \cdot x^{q'} = (r \cdot x) \cdot x^{q-1} = r \cdot x^q = x^p$$

hvor sidste lighed følger af antagelsen.

Nu kan vi altså vise invarianter, men hvad er forbindelsen til postbetingelsen, som jo er det, vi egentligt interesserer os for? Hele idéen er, at invarianten kan bruges til at vise postbetingelsen.

Hvis vi kommer til postbetingelsen, så ved vi dels, at det boolske udsagn i while-konstruktionen B må være falsk (ellers var vi jo ikke kommet ud af while-løkken), men vi ved også, at invarianten er sand, fordi vi lige var ved invariant-positionen for at checke B.

Hvis det er vist, at invarianten I gælder for en while-konstruktion, kan *partiel korrekthed* etableres ved at vise:

$$I \wedge \neg B \;\Rightarrow\; Post$$

Lad os på denne måde vise partiel korrekthed af vores eksempel-algoritme. Vi skal altså vise, at

$$r \cdot x^q = x^p \wedge \neg(q \neq 0) \;\Rightarrow\; r = x^p$$

Da $\neg(q \neq 0)$ er ensbetydende med $q = 0$, får vi ved indsættelse i $r \cdot x^q = x^p$ udsagnet $r \cdot x^0 = x^p$, hvilket er ensbetydende med $r = x^p$.

Terminering

I dette afsnit angives en skabelon beregnet på at bevise, at en while-løkke terminerer.

Idéen er at finde et eller andet, der bliver mindre efter hvert gennemløb, og som ikke kan blive vilkårligt lille. Det vil vi nu udtrykke formelt.

Vi lader $\mathcal{P}$ betegne alle variablerne i algoritmen og ønsker nu at kunne definere en funktion af disse variabler. Til det formål lader vi $\mathcal{P}_1$ betegne værdierne af alle variablerne i algoritmen første gang invariant-positionen nås. Generelt betegner $\mathcal{P}_i$ værdierne af alle variablerne den i'te gang invariant-positionen nås.

En algoritme *terminerer*, hvis der findes en *termineringsfunktion* $f\colon \mathcal{P} \to \mathbb{N}$, så

1. $\forall i \geq 1\colon f(\mathcal{P}_i) \geq 0$
2. $\forall i \geq 1\colon f(\mathcal{P}_i) > f(\mathcal{P}_{i+1})$

Intuitionen er, at da funktionen har en startværdi af en given ikke-negativ størrelse (da $f(\mathcal{P}_1) \geq 0$), og da funktionen aftager med mindst én i hvert gennemløb af while-løkken (andet krav) og ikke kan blive negativ (første krav), da må algoritmen terminere.

Invarianten, som jo allerede er vist, må naturligvis bruges til disse argumenter, hvis det er relevant.

For vores eksempel-algoritme kunne man vælge termineringsfunktionen

$$f\colon \{x, p, r, q\} \to \mathbb{N} \text{ defineret ved } f(x_i, p_i, r_i, q_i) = q_i$$

Vi verificerer nu kravene. Vi starter med det andet krav:

$$f(x_i, p_i, r_i, q_i) = q_i = q_{i+1} + 1 > q_{i+1} = f(x_{i+1}, p_{i+1}, r_{i+1}, q_{i+1})$$

Det er også klart, at det første krav er opfyldt, men vi kan faktisk ikke bevise det formelt baseret på den nuværende invariant.

For at vise det, kunne det være rart, hvis invarianten var

$$I:\ r \cdot x^q = x^p\ \wedge\ q \in \mathbb{N}$$

Så ville punktet følge direkte af deludsagnet $q \in \mathbb{N}$. Men hvis invarianten ændres, skal vi naturligvis bevise forfra, at den gælder. Lad os se på den nye del $q \in \mathbb{N}$.

Første gang vi kommer til invariant-positionen følger det af, at $q = p$, og at dette udsagn holder for p (præbetingelsen). Vi har derfor (med genbrug af det tidligere bevis for den resterende del af invarianten)

$$\boxed{p \in \mathbb{N}}\ \texttt{r = 1; q = p;}\ \boxed{r \cdot x^q = x^p \wedge q \in \mathbb{N}}$$

For de øvrige gange skal vi antage invarianten, og at betingelsen `q != 0` er sand og vise, at når kroppen er udført, er invarianten igen sand.

Men hvis $q \in \mathbb{N}$ og $q \neq 0$, da må $q \geq 1$. I kroppen udføres `q = q - 1`, så $q' = q - 1$. Det betyder klart, at $q' \geq 0$ og dermed $q' \in \mathbb{N}$.

Med genbrug af det tidligere bevis for den resterende del af invarianten har vi

$$\boxed{I \wedge q \neq 0}\ \texttt{r = r * x; q = q - 1;}\ \boxed{I}$$

hvor I er den nye invariant angivet ovenfor.

Design af invarianter

Det sidste, der mangler for at gøre behandlingen af while-løkker fuldstændig, er en række regler for, hvordan invarianter defineres. Desværre kan man ikke opstille regler, der med garanti fører frem til et bevis. Overraskende nok kan man faktisk bevise, at sådanne regler ikke kan eksistere! Formelt set er situationen faktisk endnu værere, idet der beviseligt findes algoritmer med præ- og postbetingelser, sådan at postbetingelsen altid er sand, hvis man starter i en situation, hvor præbetingelsen er sand, men hvor der simpelt hen ikke eksisterer et bevis for at det er tilfældet! De situationer er dog meget ekstreme, og man kan heldigvis komme meget langt med nogle tommelfingerregler og lidt erfaring.

Det bedste råd er at tage udgangspunkt i postbetingelsen, som jo er den, man virkelig ønsker at vise. Normalt skal man forsøge at generalisere den, så den udtaler sig om de variabler, der ændrer sig i kroppen. Samtidigt kan man holde sig for øje, at når det boolske udtryk i while-løkken bliver falsk, så skal det sammen med invarianten kunne bruges til at vise postbetingelsen.

Litteratur

[1] Sara Baase and Allen Van Gelder. *Computer Algorithms: Introduction to Design & Analysis*. Addison-Wesley, 2000.