Korrekte Software: Grundlagen und Methoden
Vorlesung 6 vom 09.05.16: Vorwärts und Rückwärts mit Floyd und Hoare.

Serge Autexier, Christoph Lüth

Universität Bremen

Sommersemester 2016

Idee

- Hier ist ein einfaches Programm:

```
// {X = x ∧ Y = y}
z = y;
// {X = x ∧ Y = z}
y = x;
// {X = y ∧ Y = z}
x = z;
// {X = y ∧ Y = x}
```

- Wir sehen:
  1. Die Verifikation erfolgt rückwärts (von hinten nach vorne).
  2. Die Verifikation kann berechnet werden.
- Muss das so sein? Ist das immer so?

Vorwärts!

- Zuweisungsregel kann rückwärts angewandt werden, weil die Nachbedingung eine offene Variable ist — P passt auf jede beliebige Nachbedingung.

$$\vdash \{P[[\![e]\!]/X]\}\, x = e\, \{P\}$$

- Alternative Zuweisungsregel (nach Floyd):

$$\frac{V \notin FV(P)}{\vdash \{P\}\, x = e\, \{\exists V.x = [\![e]\!][V/x] \wedge P[V/x]\}}$$

  - $FV(P)$ sind die freien Variablen in P.
- Jetzt ist die Vorbedingung offen— Regel kann vorwärts angewandt werden.

Vorwärtsverkettung

$$\frac{V \notin FV(P)}{\vdash \{P\}\, x = e\, \{\exists V.x = [\![e]\!][V/x] \wedge P[V/x]\}}$$

- Ein einfaches Beispiel (nach Mike Gordon):

```
// {x = 1}
x= x+1;
// {∃V.x = [[x + 1]][V/x] ∧ (x = 1)[V/x]}
```

- Vereinfachung der Nachbedingung:

$$\begin{aligned}
& \exists V.x = [\![x+1]\!][V/x] \wedge (x=1)[V/x] \\
\longleftrightarrow\ & \exists V.x = (x+1)[V/x] \wedge (x=1)[V/x] \\
\longleftrightarrow\ & \exists V.x = (V+1) \wedge (V=1) \\
\longleftrightarrow\ & x = 1+1 \\
\longleftrightarrow\ & x = 2
\end{aligned}$$

Vorwärtsverkettung

- Vorwärtsaxiom äquivalent zum Rückwärtsaxiom.
- In der Anwendung umständlicher.
- Vereinfachung benötigt Lemma:

$$\exists x.P(x) \wedge x = t \longleftrightarrow P(t)$$

- Vorteile?
  - Wir wollten doch sowieso die Anwendung automatisieren...

Schwächste Vorbedingung, stärkste Nachbedingung

- Prädikat P schwächer als Q wenn $Q \longrightarrow P$ (stärker wenn $P \longrightarrow Q$).
- Gegeben C0-Programm c, Prädikat P, dann ist
  - $\mathrm{wp}(c, P)$ die schwächste Vorbedingung Q so dass $\models \{Q\}\, c\, \{P\}$;
  - $\mathrm{sp}(P, c)$ die stärkste Nachbedingung Q so dass $\models \{P\}\, c\, \{Q\}$.
- Semantische Charakterisierung:

$$\begin{aligned}
\models \{P\}\, c\, \{Q\} &\quad\longleftrightarrow\quad P \longrightarrow \mathrm{wp}(c, Q) \\
\models \{P\}\, c\, \{Q\} &\quad\longleftrightarrow\quad \mathrm{sp}(P, c) \longrightarrow Q
\end{aligned}$$

Berechnung von $\mathrm{wp}(c, Q)$

- Einfach für Programme ohne Schleifen:

$$\begin{aligned}
\mathrm{wp}(\{\,\}, P) &\stackrel{def}{=} P \\
\mathrm{wp}(x = e, P) &\stackrel{def}{=} P[[\![e]\!]/x] \\
\mathrm{wp}(\{c\ c_s\}, P) &\stackrel{def}{=} \mathrm{wp}(c, \mathrm{wp}(\{c_s\}, P)) \\
\mathrm{wp}(\textbf{if}\ (b)\ c_0\ \textbf{else}\ c_1, P) &\stackrel{def}{=} ([\![b]\!] \wedge \mathrm{wp}(c_0, P)) \vee (\neg [\![b]\!] \wedge \mathrm{wp}(c_1, P))
\end{aligned}$$

- Für Schleifen: nicht entscheidbar.
  - "Cannot in general compute a finite formula" (Gordon)
- Wir können rekursive Formulierung angeben:

$$\mathrm{wp}(\textbf{while}\ (b)\ \{c\}, P) \stackrel{def}{=} (\neg [\![b]\!] \wedge P) \vee ([\![b]\!] \wedge \mathrm{wp}(c, \mathrm{wp}(\textbf{while}\ (b)\ \{c\}, P)))$$

  - Hilft auch nicht weiter...

Lösung: Annotierte Programme

- Wir helfen dem Rechner weiter und annotieren die Schleifeninvariante am Programm.
- Damit berechnen wir:
  - die approximative schwächste Vorbedingung $\mathrm{awp}(c, Q)$ zusammen mit einer Menge von Verifikationsbedingungen $\mathrm{wvc}(c, Q)$
  - oder die approximative stärkste Nachbedingung $\mathrm{asp}(P, c)$ zusammen mit einer Menge von Verifikationsbedingungen $\mathrm{svc}(P, c)$
- Es gilt:

$$\begin{aligned}
\textstyle\bigwedge \mathrm{wvc}(c, Q) &\longrightarrow \models \{\mathrm{awp}(c, Q)\}\, c\, \{Q\} \\
\textstyle\bigwedge \mathrm{svc}(P, c) &\longrightarrow \models \{P\}\, c\, \{\mathrm{asp}(P, c)\}
\end{aligned}$$

Approximative schwächste Vorbedingung

$$
\begin{aligned}
\mathrm{awp}(\{\,\},P) &\stackrel{def}{=} P\\
\mathrm{awp}(x=e,P) &\stackrel{def}{=} P[[\![e]\!]/x]\\
\mathrm{awp}(\{c\ c_s\},P) &\stackrel{def}{=} \mathrm{awp}(c,\mathrm{awp}(\{c_s\},P))\\
\mathrm{awp}(\textbf{if}\ (b)\ c_0\ \textbf{else}\ c_1,P) &\stackrel{def}{=} (b\wedge\mathrm{awp}(c_0,P))\vee(\neg b\wedge\mathrm{awp}(c_1,P))\\
\mathrm{awp}(/{*}{*}\ \{q\}\ {*}/,P) &\stackrel{def}{=} [\![q]\!]\\
\mathrm{awp}(\textbf{while}\ (b)\ /{*}{*}\ \textbf{inv}\ i\ {*}/\ c,P) &\stackrel{def}{=} [\![i]\!]
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{wvc}(\{\,\},P) &\stackrel{def}{=} \emptyset\\
\mathrm{wvc}(x=e,P) &\stackrel{def}{=} \emptyset\\
\mathrm{wvc}(\{c\ c_s\},P) &\stackrel{def}{=} \mathrm{wvc}(c,\mathrm{awp}(\{c_s\},P))\cup\mathrm{wvc}(\{c_s\},P)\\
\mathrm{wvc}(\textbf{if}\ (b)\ c_0\ \textbf{else}\ c_1,P) &\stackrel{def}{=} \mathrm{wvc}(c_0,P)\cup\mathrm{wvc}(c_1,P)\\
\mathrm{wvc}(/{*}{*}\ \{q\}\ {*}/,P) &\stackrel{def}{=} \{[\![q]\!]\longrightarrow P\}\\
\mathrm{wvc}(\textbf{while}\ b\ /{*}{*}\ \textbf{inv}\ i\ {*}/\ c,P) &\stackrel{def}{=} \mathrm{wvc}(c,[\![i]\!])\\
&\quad\cup\{[\![i]\!]\wedge b\longrightarrow\mathrm{awp}(c,[\![i]\!])\}\\
&\quad\cup\{[\![i]\!]\wedge\neg b\longrightarrow P\}
\end{aligned}
$$

Approximative stärkste Nachbedingung

$$
\begin{aligned}
\mathrm{asp}(P,\{\,\}) &\stackrel{def}{=} P\\
\mathrm{asp}(P,x=e) &\stackrel{def}{=} \exists V.x=[\![e]\!][V/x]\wedge P[V/x]\\
\mathrm{asp}(P,\{c\ c_s\}) &\stackrel{def}{=} \mathrm{asp}(\mathrm{asp}(P,c),c_s)\\
\mathrm{asp}(P,\textbf{if}\ (b)\ c_0\ \textbf{else}\ c_1,P) &\stackrel{def}{=} \mathrm{asp}([\![b]\!]\wedge P,c_0)\vee\mathrm{asp}(\neg[\![b]\!]\wedge P,c_1)\\
\mathrm{asp}(P,/{*}{*}\ \{q\}\ {*}/) &\stackrel{def}{=} [\![q]\!]\\
\mathrm{asp}(P,\textbf{while}\ (b)\ /{*}{*}\ \textbf{inv}\ i\ {*}/\ c) &\stackrel{def}{=} [\![i]\!]\wedge\neg[\![b]\!]
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{svc}(P,\{\,\}) &\stackrel{def}{=} \emptyset\\
\mathrm{svc}(P,x=e) &\stackrel{def}{=} \emptyset\\
\mathrm{svc}(P,\{c\ c_s\}) &\stackrel{def}{=} \mathrm{svc}(P,c)\cup\mathrm{svc}(\mathrm{asp}(P,c),\{c_s\})\\
\mathrm{svc}(P,\textbf{if}\ (b)\ c_0\ \textbf{else}\ c_1) &\stackrel{def}{=} \mathrm{svc}([\![b]\!]\wedge P,c_0)\cup\mathrm{svc}(\neg[\![b]\!]\wedge P,c_1)\\
\mathrm{svc}(P,/{*}{*}\ \{q\}\ {*}/) &\stackrel{def}{=} \{P\longrightarrow[\![q]\!]\}\\
\mathrm{svc}(P,\textbf{while}\ b\ /{*}{*}\ \textbf{inv}\ i\ {*}/\ c) &\stackrel{def}{=} \{P\longrightarrow[\![i]\!]\}\\
&\quad\cup\{\mathrm{asp}([\![b]\!]\wedge[\![i]\!],c)\longrightarrow[\![i]\!]\}\\
&\quad\cup\mathrm{svc}([\![b]\!]\wedge[\![i]\!],c)
\end{aligned}
$$

Beispiel: das Fakultätsprogramm

- In der Praxis sind Vor- und Nachbedingung gegeben, und nur die Verifikationsbedingungen relevant:

Sei F das annotierte Fakultätsprogramm:

```
// {0 <= N}
p= 1;
c= 1;
while (c <= n) /** inv p == fac(c-1) && c-1 <= N; */ {
  p = p * c;
  c = c + 1;
}
// {p == fac(N)}
```

- Berechnung vorwärts: $\mathrm{svc}(F,\top)$
- Berechnung rückwärts: $\mathrm{wvc}(\top,F)$

Zusammenfassung

- Die Zuweisungsregel gibt es"rückwärts" und "vorwärts".
- Bis auf die Invarianten an Schleifen können wir Korrektheit automatisch prüfen.

Rückwärtsberechnung:
- Einfacher zu berechnen
- Führt zu großen Formeln
- Keine Möglichkeit, Zwischenzustände zu vereinfachen

Vorwärtsberechnung:
- Entspricht symbolischer Ausführung
- Umständlichere Berechnung der Verifikationsbedingungen
- Erlaubt zustandsbasierte Vereinfachung (z.B. Entfernen unerreichbarer Fälle)

- Die Generierung von Verifikationsbedingungen korrespondiert zur relativen Vollständigkeit der Floyd-Hoare-Logik — nächste Woche.