Von Z zu Code

Andreas Zeller
Lehrstuhl für Softwaretechnik
Universität des Saarlandes, Saarbrücken

2005-12-01

Übersicht

- Verfeinerung (Reïfikation)
- Axiomatische Ableitung
- Von Z zum Code

Verfeinerung

Verfeinerung = Übergang vom *abstrakten* zum *konkreteren*:

Daten-Verfeinerung Angabe einer konkreten Darstellung für Daten (z.B. Liste statt Menge)

Operations-Verfeinerung Angabe von konkreten „Implementierungen“ zu abstrakten Operationen

Auch bekannt als *Reïfikation* (*Versachlichung*)

Was ist Verfeinerung?

Zentrale Eigenschaft von Verfeinerungen:

Die konkretere Spezifikation muss die Eigenschaften der abstrakteren Spezifikation erfüllen

Beispiel – Die abstrakte Spezifikation verlangt

$$x' > x$$

Konkreteres Prädikat (z.B. aus der Implementierung):

$$x' = x + 1$$

Beweisverpflichtung (trivial):

$$x' = x + 1 \Rightarrow x' > x$$

Telefonbuch – abstrakt

Wir definieren ein einfaches Telefonbuch:

$$[NAME, PHONE]$$

$$\begin{array}{|l}
\hline
\textit{PhoneBook} \\
\textit{entries} : \textit{NAME} \rightarrowtail \textit{PHONE} \\
\hline
\end{array}$$

$$\begin{array}{|l}
\hline
\textit{InitPhoneBook} \\
\textit{PhoneBook} \\
\hline
\textit{entries} = \varnothing \\
\hline
\end{array}$$

Hinzufügen – abstrakt

Wir definieren ein Operations-Schema, das einen neuen Namen und Nummer hinzufügt:

$$\begin{array}{|l}
\hline
\textit{AddPhone} \\
\Delta \textit{PhoneBook} \\
n? : \textit{NAME} \\
ph? : \textit{PHONE} \\
\hline
n? \notin \operatorname{dom} \textit{entries} \\
\textit{entries}' = \textit{entries} \cup \{n? \mapsto ph?\} \\
\hline
\end{array}$$

Telefonbuch – konkret

Die konkrete Realisierung benutzt eine Liste von Verbünden:

$$\begin{array}{l} \text{Entry} \\ \hline name : NAME \\ phone : PHONE \end{array}$$

wobei gilt

$$\forall e_1, e_2 : Entry \bullet e_1.name = e_2.name \Leftrightarrow e_1 = e_2$$

$$\begin{array}{l} \text{PhoneBook\_1} \\ \hline entries\_1 : \operatorname{seq} Entry \end{array}$$

$$\begin{array}{l} \text{InitPhoneBook\_1} \\ \hline entries\_1 = \langle\,\rangle \end{array}$$

Hinzufügen – konkret

Das Hinzufügen eines neuen Namen und Nummer geschieht über das Anhängen eines neuen Verbundes:

$$\begin{array}{l} \text{AddPhone\_1} \\ \hline \Delta PhoneBook\_1 \\ n? : NAME \\ ph? : PHONE \\ \hline \neg(\exists e : Entry \mid e \in \operatorname{ran} entries\_1 \bullet e.name = n?) \\ entries\_1' = entries\_1 \frown \\ \qquad \langle(\mu\, e : Entry \mid e.name = n? \wedge e.phone = ph?)\rangle \end{array}$$

μ: Konstruktor des *Entry*-Verbundes

Verfeinerungs-Schema

Das Verfeinerungs-Schema (auch *Abstraktions-Schema*) gibt an, wie konkrete Zustände mit abstrakten Zuständen zusammenhängen:

$$\begin{array}{l} \text{Refine} \\ \hline PhoneBook \\ PhoneBook\_1 \\ \hline \operatorname{dom} entries = \{e : Entry \mid e \in \operatorname{ran} entries\_1 \bullet e.name\} \\ \forall i \in \{1, \ldots, \#entries\_1\} \bullet \\ \qquad entries\_1(i).phone = entries(entries\_1(i).name) \end{array}$$

Beweisverpflichtungen

Die konkretere Spezifikation muss die Eigenschaften der abstrakteren Spezifikation erfüllen

Bei Daten-Verfeinerung muss bewiesen werden:

Gültiger Ursprungszustand. Jeder Ursprungszustand im Konkreten muss einem Ursprungszustand im Abstrakten entsprechen.

Schwächere Vorbedingung. Die Vorbedingung der konkreten Operation muss schwächer sein als die der abstrakten.

Stärkere Nachbedingung. Die Nachbedingung der konkreten Operation muss stärker sein als die der abstrakten.

(Vergleiche: Verfeinerte Klassen bei Vererbung!)

Gültiger Ursprungszustand

Jeder Ursprungszustand im Konkreten muss einem Ursprungszustand im Abstrakten entsprechen.

Formal:

$$\forall\, AbsState;\ ConState \bullet ConInit \land Refine \Rightarrow AbsInit$$

$AbsState$: Abstrakter Zustandsraum
$ConState$: Konkreter Zustandsraum
$ConInit$: Ursprungszustand im Konkreten
$Refine$: Verfeinerungs-Schema von $AbsState$ nach $ConState$
$AbsInit$: Ursprungszustand im Abstrakten

Gültiger Ursprungszustand – Telefonbuch

Beweisverpflichtung:

$$\forall\, PhoneBook;\ PhoneBook\_1 \bullet \\ \quad InitPhoneBook\_1 \land Refine \Rightarrow InitPhoneBook$$

Wir expandieren die linke Seite der Implikation:

$$entries\_1 = \langle\,\rangle \land \\ \operatorname{dom} entries = \{e : Entry \mid e \in \operatorname{ran} entries\_1 \bullet e.name\} \land \\ \forall\, i \in \{1, \ldots, \#entries\_1\} \bullet \\ \quad entries\_1(i).phone = entries(entries\_1(i).name)$$

Wegen $entries\_1 = \langle\,\rangle$ gilt:

$$\{e : Entry \mid e \in \operatorname{ran} entries\_1 \bullet e.name\} = \varnothing \qquad \text{und} \\ \operatorname{dom} entries = entries = \varnothing$$

womit das Prädikat von $InitPhoneBook$ gezeigt wäre.

Schwächere Vorbedingung

Die Vorbedingung der konkreten Operation muss schwächer sein als die der abstrakten.

Formal:

$$\forall\, AbsState;\; ConState;\; x? : X \bullet \\ \quad \mathrm{pre}\, AbsOp \land Refine \Rightarrow \mathrm{pre}\, ConOp$$

$x? : X$: Eingabeparameter der Operation
$AbsOp$: Operation im Abstrakten
$ConOp$: Operation im Konkreten
pre: Vorbedingung

Schwächere Vorbedingung – Telefonbuch

Beweisverpflichtung:

$$\forall\, PhoneBook;\; PhoneBook\_1;\; n? : NAME;\; ph? : PHONE \bullet \\ \quad \mathrm{pre}\, AddPhone \land Refine \Rightarrow \mathrm{pre}\, AddPhone\_1$$

Expandiert (nach Berechnung der Vorbedingungen):

$$n? \notin \mathrm{dom}\, entries \land \\ \mathrm{dom}\, entries = \{e : Entry \mid e \in \mathrm{ran}\, entries\_1 \bullet e.name\} \land \\ (\forall\, i \in \{1, \ldots, \#entries\_1\} \bullet \\ \quad entries\_1(i).phone = entries(entries\_1(i).name)) \Rightarrow \\ \quad\quad \neg(\exists\, e : Entry \mid e \in \mathrm{ran}\, entries \bullet e.name = n?)$$

Schwächere Vorbedingung (2)

Aus

$$n? \notin \mathrm{dom}\, entries \land \\ \mathrm{dom}\, entries = \{e : Entry \mid e \in \mathrm{ran}\, entries\_1 \bullet e.name\}$$

schließen wir

$$n? \notin \{e : Entry \mid e \in \mathrm{ran}\, entries\_1 \bullet e.name\}$$

Dies lässt sich umschreiben zu

$$\forall\, e : Entry \mid e \in \mathrm{ran}\, entries\_1 \bullet e.name \neq n?$$

oder gleich

$$\neg(\exists\, e : Entry \mid e \in \mathrm{ran}\, entries \bullet e.name = n?)$$

was zu zeigen war.

Stärkere Nachbedingung

Die Nachbedingung der konkreten Operation muss stärker sein als die der abstrakten.

Formal:

$$\begin{array}{l} \forall\, \Delta AbsState;\ \Delta ConState;\ x? : X;\ y! : Y \bullet \\ \quad \mathrm{pre}\, AbsOp \land ConOp \land \Delta Refine \Rightarrow AbsOp \end{array}$$

Das Verfeinerungs-Schema gilt hier für alle Zustände:

$$\begin{array}{ccc} AbsState & \xrightarrow{AbsOp} & AbsState' \\ {\scriptstyle Abs}\uparrow & & \uparrow{\scriptstyle Abs} \\ ConState & \xrightarrow[ConOp]{} & ConState' \end{array}$$

Stärkere Nachbedingung – Telefonbuch

Beweisverpflichtung:

$$\begin{array}{l} \forall\, \Delta PhoneBook;\ \Delta PhoneBook\_1;\ n? : NAME;\ ph? : PHONE \bullet \\ \quad \mathrm{pre}\, AddPhone \land AddPhone\_1 \land \Delta Refine \Rightarrow AddPhone \end{array}$$

Expandiert:

$$\begin{array}{l} n? \notin \mathrm{dom}\, entries \land \\ \mathrm{dom}\, entries = \{ e : Entry \mid e \in \mathrm{ran}\, entries\_1 \bullet e.name \} \land \\ (\forall\, i \in \{1, \ldots, \# entries\_1\} \bullet \\ \quad entries\_1(i).phone = entries(entries\_1(i).name)) \land \\ \neg (\exists\, e : Entry \mid e \in \mathrm{ran}\, entries\_1 \bullet e.name = n?) \land \\ entries\_1' = entries\_1 ^\frown \\ \quad \langle (\mu\, e : Entry \mid e.name = n? \land e.phone = ph?) \rangle \land \\ \mathrm{dom}\, entries' = \{ e : Entry \mid e \in \mathrm{ran}\, entries\_1' \bullet e.name \} \land \\ (\forall\, i \in \{1, \ldots, \# entries\_1'\} \bullet \\ \quad entries\_1'(i).phone = entries'(entries\_1'(i).name)) \Rightarrow \\ \quad (n? \notin \mathrm{dom}\, entries \land entries' = entries \cup \{ n? \mapsto ph? \}) \end{array}$$

Stärkere Nachbedingung (2)

Zu zeigen:

$$
\begin{array}{l}
n? \notin \operatorname{dom} entries \land \\
\operatorname{dom} entries = \{e : Entry \mid e \in \operatorname{ran} entries\_1 \bullet e.name\} \land \\
(\forall i \in \{1, \ldots, \#entries\_1\} \bullet \\
\qquad entries\_1(i).phone = entries(entries\_1(i).name)) \land \\
\neg(\exists e : Entry \mid e \in \operatorname{ran} entries\_1 \bullet e.name = n?) \land \\
entries\_1' = entries\_1 ^\frown \\
\qquad \langle(\mu\, e : Entry \mid e.name = n? \land e.phone = ph?)\rangle \land \\
\operatorname{dom} entries' = \{e : Entry \mid e \in \operatorname{ran} entries\_1' \bullet e.name\} \land \\
(\forall i \in \{1, \ldots, \#entries\_1'\} \bullet \\
\qquad entries\_1'(i).phone = entries'(entries\_1'(i).name)) \Rightarrow \\
\qquad (n? \notin \operatorname{dom} entries \land entries' = entries \cup \{n? \mapsto ph?\})
\end{array}
$$

Wir müssen nur $entries' = entries \cup \{n? \mapsto ph?\}$ zeigen – durch (1) $n? \in \operatorname{dom} entries'$ und (2) $entries'(n?) = ph?$

Stärkere Nachbedingung (3)

Ziel: Zeigen, dass (1) $n? \in \operatorname{dom} entries'$ gilt.

Wir betrachten das Prädikat

$$\operatorname{dom} entries' = \{e : Entry \mid e \in \operatorname{ran} entries\_1' \bullet e.name\}$$

Hier ist $entries\_1'$

$$entries\_1' = entries\_1 {}^\frown \langle (\mu\, e_1 : Entry \mid e_1.name = n? \wedge e_1.phone = ph?) \rangle$$

Folgerung: Es gibt in $entries\_1'$ einen Eintrag e_1 mit Namen $n?$.

Folge: $(n?, ph?) \in \operatorname{ran} entries'$ und somit $n? \in \operatorname{dom} entries'$.

Stärkere Nachbedingung (4)

Ziel: Zeigen, dass (2) $entries'(n?) = ph?$ gilt.

Aus (1) $n? \in \operatorname{dom} entries'$ und dem Prädikat

$$(\forall\, i \in \{1, \ldots, \#entries\_1'\} \bullet entries\_1'(i).phone = entries'(entries\_1'(i).name))$$

schließen wir, dass (A)

$$entries'(n?) = \{i \in \{1, \ldots, \#entries\_1'\} \mid entries\_1'(i).name = n? \bullet entries\_1'(i).phone\}$$

Aus dem Prädikat

$$entries\_1' = entries\_1 {}^\frown \langle (\mu\, e_1 : Entry \mid e_1.name = n? \wedge e_1.phone = ph?) \rangle$$

wissen wir, dass (B)

$$\exists\, i \in \{1, \ldots, \#entries\_1'\} \bullet e.name = n? \wedge e.phone = ph?$$

Das Ziel (2) $entries'(n?) = ph?$ folgt aus (A) und (B).

Beweisverpflichtungen – Telefonbuch

Wir haben gezeigt, dass

- Jeder Ursprungszustand des konkreten Telefonbuchs einem Ursprungszustand des abstrakten Telefonbuchs entspricht
- Die Vorbedingung der konkreten Operation schwächer ist als die der abstrakten
- Die Nachbedingung der konkreten Operation stärker als die der abstrakten

Folge: *Das konkrete Telefonbuch erfüllt alle Eigenschaften des abstrakten Telefonbuchs* (was zu zeigen war).

Von Z zum Code

Ziel: Aus Spezifikation korrekten Code erzeugen.

Verfahren:

Axiomatische Ableitung. Entwicklung des Programms aus Vor- und Nachbedingungen (Hoare-Kalkül)

Verfeinerungs-Kalkül. Abbildung der Spezifikationssprache auf die Programmiersprache

Axiomatische Methode

Grund-Idee: *Hoare-Tripel*

$$\{P\}\ S\ \{Q\}$$

Wenn das Programm in Zustand P ist und S ausführt, ist es anschließend in Zustand Q.

Beispiel: $iroot$

$$\begin{array}{|l}
iroot : \mathbb{N} \to \mathbb{N} \\
\hline
\forall\, a : \mathbb{N} \bullet (\mathbf{let}\, r == iroot(a) \bullet \\
\qquad\qquad a \geq 0 \wedge \\
\qquad\qquad 0 \leq r * r \leq a < (r+1) * (r+1))
\end{array}$$

Hoare-Tripel:

$$\{a \geq 0\}\ R\ \{0 \leq r * r \leq a < (r+1) * (r+1))\}$$

Axiomatische Methode (2)

Wir möchten $iroot$ mit Hilfe einer Schleife berechnen:
Wir erhöhen r, bis die Nachbedingung erfüllt ist.

Axiomatische Definition der `while`-Anweisung:

$$\begin{array}{l}
\{I\} \\
\texttt{while}(p)\{p \wedge I\}S\{I' \wedge v' < v\} \\
\{\neg\, p \wedge I\}
\end{array}$$

p: Wächter
v: Variante (ganze Zahl)
I: Invariante

Axiomatische Methode (3)

Wir instanziieren die axiomatische Definition der `while`-Anweisung:

$\{a \geq 0\}$
`r = 0;`
$\{I\}$
`while` (p) $\{p \wedge I\}$`r = r + 1;`$\{I' \wedge v' < v\}$
$\{\neg p \wedge I\}$
$\{0 \leq r * r \leq a < (r+1) * (r+1))\}$

$\{\neg p \wedge I\}$ muss unser Ziel sein – wir wählen also

- als Wächter p die Negation des Prädikats: $a \geq (r+1) * (r+1)$
- als Invariante I das Prädikat: $0 \leq r * r \leq a$

Axiomatische Methode (4)

Wir erhalten

$\{a \geq 0\}$
`r = 0;`
$\{0 \leq r * r \leq a\}$
`while` $(a \geq (r+1) * (r+1))$ `r = r + 1;` $\{v' < v\}$
$\{a < (r+1) * (r+1) \wedge 0 \leq r * r \leq a\}$
$\{0 \leq r * r \leq a < (r+1) * (r+1))\}$

Die beiden letzten Prädikate sind äquivalent
$\Rightarrow$ die Verifikation ist gelungen.

Um zu zeigen, dass die Schleife terminiert, setzen wir v auf die verbleibende Differenz zwischen a und dem Quadrat von r, also $v = a - (r+1) * (r+1)$.

Axiomatische Methode (5)

In der Praxis funktioniert die axiomatische Methode am besten,

- wenn Prädikate und Code *gemeinsam entwickelt werden* (Korrektheit folgt „automatisch“ aus der Verfeinerung)
- wenn Beweise maschinengestützt erzeugt und geprüft werden können (etwa mit Hilfe eines Theorembeweisers)

Mehr dazu: Vorlesungen zum Thema Programmverifikation

Verfeinerungs-Kalkül

Verfeinerungs-Kalkül = *Abbildung der Spezifikationssprache auf die Programmiersprache*

Grundlage: *Verfeinerungsregeln* der Form

Spezifikations-Ausdruck $\sqsubseteq$ Ausdruck der Programmiersprache

Hierbei bedeutet $\sqsubseteq$: „übersetzt nach“ oder „wird realisiert durch“

Einfache Beispiele – Z nach C:

$\mathit{false} \sqsubseteq 0$	[C false]
$(p \land q) \lor (\neg p \land r) \sqsubseteq p\,?\,q : r$	[Bedingtes Prädikat]

Verfeinerungs-Kalkül (2)

Beispiel: Wie wird $p \land q$ ausgewertet?

$p \land q$	[gegeben]
$\Leftrightarrow (p \land q) \lor \mathit{false}$	[$p \lor \mathit{false} \Leftrightarrow p$]
$\Leftrightarrow (p \land q) \lor (\neg p \land \mathit{false})$	[$p \land \mathit{false} \Leftrightarrow \mathit{false}$)]
$\sqsubseteq p\,?\,q : \mathit{false}$	[Bedingtes Prädikat]
$\sqsubseteq p\,?\,q : 0$	[C false]

Daten verfeinern

Die meisten Daten in Z können direkt in herkömmliche Datenstrukturen verfeinert werden:

- Mengen werden zu Feldern oder Bäumen: $x \in s \sqsubseteq$ `s[x]`
- Folgen werden zu Listen oder Feldern: $\mathit{head}\ s \sqsubseteq$ `s[0]`
- Abbildungen werden zu Hashtabellen oder Bäumen:
  $\mathit{entries}(i) \sqsubseteq$ `entries(i)`

Zustands-Schemata verfeinern

Zustands-Schemata können zu *Typen* verfeinert werden:

Entry
$\mathit{name} : \mathit{NAME}$
$\mathit{phone} : \mathit{PHONE}$

wird in C zu

```
typedef struct {
    name: NAME;
    phone: PHONE;
} Entry;
```

Zuweisung verfeinern

Die einfachste Verfeinerung ist die *unveränderte Variable:*

$x' = x \quad \sqsubseteq \quad \langle$leere Anweisung$\rangle$

Ansonsten eher trivial:

$x' = e \quad \sqsubseteq$ `x' = e`

Mehrfache Zuweisung kann aber haariger werden:

$x' = y \wedge y' = x \quad \sqsubseteq$ `t = x; x = y; y = t`

Spezielle Regeln für *komplexere Datenstrukturen* – etwa:

$S' = S \cup \{x\} \quad \sqsubseteq$ `s[x] = 1`

Bedingungen

Typisch: *Bedingter Zustandswechsel*

$p \wedge s \quad \sqsubseteq$ `if (p) s`

Disjunktion wird zu *Fallunterscheidung:*

$(p \wedge s) \vee (q \wedge t) \quad \sqsubseteq$ `if (p) s; else if (q) t`

– setzt voraus, dass p und q disjunkt sind!

Konjunktionen

Konjunktionen können in der Regel nicht unmittelbar übersetzt werden.

Beispiel – Division:

$$Quotient \mathrel{\widehat{=}} [n, d, q, r : \mathbb{Z} \mid d \neq 0 \wedge n = q * d + r]$$
$$Remainder \mathrel{\widehat{=}} [r, d : \mathbb{Z} \mid r < d]$$

Mögliche Verfeinerungen:

$d \neq 0 \wedge n = q * d + r \quad \sqsubseteq$ `q = 0; r = n`
$r < d \quad \sqsubseteq$ `r = 0;`

Können wir daraus eine Verfeinerung für $Division \mathrel{\widehat{=}} Quotient \wedge Remainder$ bilden?

Nein – es geht nur konstruktiv: etwa
`for (q = 0, r = n; r >= d; q++) r = r - d;`

Quantoren

Quantoren sind leicht zu realisieren:

$$\forall\, x : S \bullet p(x) \;\sqsubseteq\; \texttt{b = 1; for}(x \in S)\texttt{if}(!p(x))\texttt{b = 0;}$$
$$\exists\, x : S \bullet p(x) \;\sqsubseteq\; \texttt{b = 0; for}(x \in S)\texttt{if}(p(x))\texttt{b = 1;}$$

Der Ausdruck $x \in S$ muss abhängig von der Mengen-Darstellung verfeinert werden.

Standard in *ausführbaren Spezifikationen*!

Operations-Schemata

werden zu Prozeduren auf globalen Variablen oder Typen

$$S \mathrel{\widehat{=}} [x, y : \mathbb{Z}]$$
$$Op \mathrel{\widehat{=}} [\Delta S \mid x' = x + y;\ ]$$

verfeinert zu

```
int x, y;
void op(void) { x = x + y; }
```

oder auch

```
typedef struct { int x, y; } S;
void op(S* s) { s->x = s->x + s->y; }
```

Schema-Ausdrücke

Ein *Gesamt-Schema* der Art

$$Main \mathrel{\widehat{=}} Op_1 \vee Op_2 \vee \ldots \vee Op_n \vee Exception$$

verfeinert zu einem *ereignisgesteuerten Programm:*

```
while (ok) {
    get_event();
    if (test_1()) do_1();
    else if (test_2()) do_2();
    ...
    else if (test_n()) do_n();
    else exception()
}
```

Zusammenfassung

- Verfeinerung: Übergang vom abstrakten zum konkreten
- Die konkretere Spezifikation muss die Eigenschaften der abstrakteren Spezifikation erfüllen:
  - Gültiger Ursprungszustand
  - Schwächere Vorbedingung
  - Stärkere Nachbedingung
- Von einer Z-Spezifikation erhält man Code
  - per axiomatischer Ableitung oder
  - per Verfeinerungs-Kalkül
- Verfeinerung stellt sicher, dass der endgültige Programmcode die zugesicherten Eigenschaften erfüllt