

Von Z zu Code

Andreas Zeller
Lehrstuhl für Softwaretechnik
Universität des Saarlandes, Saarbrücken
2005-12-01

Übersicht

- Verfeinerung (Reifikation)
- 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 *Reifikation* (*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:

$[NAME, PHONE]$

<i>PhoneBook</i> $entries : NAME \rightarrow PHONE$
--

<i>InitPhoneBook</i> <i>PhoneBook</i> $entries = \emptyset$

Hinzufügen – abstrakt

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

<i>AddPhone</i> $\Delta PhoneBook$ $n? : NAME$ $ph? : PHONE$ $n? \notin \text{dom } entries$ $entries' = entries \cup \{n? \mapsto ph?\}$
--

Telefonbuch – konkret

Die konkrete Realisierung benutzt eine Liste von Verbünden:

<i>Entry</i> <i>name</i> : <i>NAME</i> <i>phone</i> : <i>PHONE</i>
--

wobei gilt

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

<i>PhoneBook_1</i> <i>entries_1</i> : seq <i>Entry</i>

<i>InitPhoneBook_1</i> <i>entries_1</i> = $\langle \rangle$
--

Hinzufügen – konkret

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

<i>AddPhone_1</i> $\Delta \text{PhoneBook}_1$ <i>n?</i> : <i>NAME</i> <i>ph?</i> : <i>PHONE</i>
$\neg(\exists e : \text{Entry} \mid e \in \text{ran } \text{entries}_1 \bullet e.\text{name} = n?)$ $\text{entries}_1' = \text{entries}_1 \hat{\ } \langle (\mu e : \text{Entry} \mid e.\text{name} = n? \wedge e.\text{phone} = \text{ph?}) \rangle$

μ : Konstruktor des *Entry*-Verbundes

Verfeinerungs-Schema

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

<i>Refine</i> <i>PhoneBook</i> <i>PhoneBook_1</i>
$\text{dom } \text{entries} = \{e : \text{Entry} \mid e \in \text{ran } \text{entries}_1 \bullet e.\text{name}\}$ $\forall i \in \{1, \dots, \# \text{entries}_1\} \bullet$ $\text{entries}_1(i).\text{phone} = \text{entries}(\text{entries}_1(i).\text{name})$

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:

$$\forall \text{ AbsState}; \text{ ConState} \bullet \text{ ConInit} \wedge \text{Refine} \Rightarrow \text{AbsInit}$$

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:

$$\forall \text{ PhoneBook}; \text{ PhoneBook}_1 \bullet \\ \text{InitPhoneBook}_1 \wedge \text{Refine} \Rightarrow \text{InitPhoneBook}$$

Wir expandieren die linke Seite der Implikation:

$$\begin{aligned} \text{entries}_1 &= \langle \rangle \wedge \\ \text{dom entries} &= \{e : \text{Entry} \mid e \in \text{ran entries}_1 \bullet e.\text{name}\} \wedge \\ \forall i \in \{1, \dots, \#\text{entries}_1\} \bullet \\ &\quad \text{entries}_1(i).\text{phone} = \text{entries}(\text{entries}_1(i).\text{name}) \end{aligned}$$

Wegen $\text{entries}_1 = \langle \rangle$ gilt:

$$\{e : \text{Entry} \mid e \in \text{ran entries}_1 \bullet e.\text{name}\} = \emptyset \quad \text{und} \\ \text{dom entries} = \text{entries} = \emptyset$$

womit das Prädikat von *InitPhoneBook* gezeigt wäre.

Schwächere Vorbedingung

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

Formal:

$$\forall \text{ AbsState; ConState; } x? : X \bullet \\ \text{pre AbsOp} \wedge \text{Refine} \Rightarrow \text{pre ConOp}$$

$x? : X$: Eingabeparameter der Operation

AbsOp : Operation im Abstrakten

ConOp : Operation im Konkreten

pre: Vorbedingung

Schwächere Vorbedingung – Telefonbuch

Beweisverpflichtung:

$$\forall \text{ PhoneBook; PhoneBook_1; } n? : \text{NAME; } ph? : \text{PHONE} \bullet \\ \text{pre AddPhone} \wedge \text{Refine} \Rightarrow \text{pre AddPhone_1}$$

Expandiert (nach Berechnung der Vorbedingungen):

$$\begin{aligned} & n? \notin \text{dom entries} \wedge \\ & \text{dom entries} = \{e : \text{Entry} \mid e \in \text{ran entries_1} \bullet e.\text{name}\} \wedge \\ & (\forall i \in \{1, \dots, \#\text{entries_1}\} \bullet \\ & \quad \text{entries_1}(i).\text{phone} = \text{entries}(\text{entries_1}(i).\text{name})) \Rightarrow \\ & \quad \neg(\exists e : \text{Entry} \mid e \in \text{ran entries} \bullet e.\text{name} = n?) \end{aligned}$$

Schwächere Vorbedingung (2)

Aus

$$\begin{aligned} & n? \notin \text{dom entries} \wedge \\ & \text{dom entries} = \{e : \text{Entry} \mid e \in \text{ran entries_1} \bullet e.\text{name}\} \end{aligned}$$

schließen wir

$$n? \notin \{e : \text{Entry} \mid e \in \text{ran entries_1} \bullet e.\text{name}\}$$

Dies lässt sich umschreiben zu

$$\forall e : \text{Entry} \mid e \in \text{ran entries_1} \bullet e.\text{name} \neq n?$$

oder gleich

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

was zu zeigen war.

Stärkere Nachbedingung

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

Formal:

$$\forall \Delta AbsState; \Delta ConState; x? : X; y! : Y \bullet \\ \text{pre } AbsOp \wedge ConOp \wedge \Delta Refine \Rightarrow AbsOp$$

Das Verfeinerungs-Schema gilt hier für alle Zustände:

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

Stärkere Nachbedingung – Telefonbuch

Beweisverpflichtung:

$$\forall \Delta PhoneBook; \Delta PhoneBook_1; n? : NAME; ph? : PHONE \bullet \\ \text{pre } AddPhone \wedge AddPhone_1 \wedge \Delta Refine \Rightarrow AddPhone$$

Expandiert:

$$\begin{aligned} & n? \notin \text{dom } entries \wedge \\ & \text{dom } entries = \{e : Entry \mid e \in \text{ran } entries_1 \bullet e.name\} \wedge \\ & (\forall i \in \{1, \dots, \#entries_1\} \bullet \\ & \quad entries_1(i).phone = entries(entries_1(i).name)) \wedge \\ & \neg(\exists e : Entry \mid e \in \text{ran } entries \bullet e.name = n?) \wedge \\ & entries_1' = entries_1 \wedge \\ & \quad \langle (\mu e : Entry \mid e.name = n? \wedge e.phone = ph?) \rangle \wedge \\ & \text{dom } entries' = \{e : Entry \mid e \in \text{ran } entries_1' \bullet e.name\} \wedge \\ & (\forall i \in \{1, \dots, \#entries_1'\} \bullet \\ & \quad entries_1'(i).phone = entries'(entries_1'(i).name)) \Rightarrow \\ & (n? \notin \text{dom } entries \wedge entries' = entries \cup \{n? \mapsto ph?\}) \end{aligned}$$

Stärkere Nachbedingung (2)

Zu zeigen:

$$\begin{aligned} & n? \notin \text{dom } \text{entries} \wedge \\ & \text{dom } \text{entries} = \{e : \text{Entry} \mid e \in \text{ran } \text{entries_1} \bullet e.\text{name}\} \wedge \\ & (\forall i \in \{1, \dots, \#\text{entries_1}\} \bullet \\ & \quad \text{entries_1}(i).\text{phone} = \text{entries}(\text{entries_1}(i).\text{name})) \wedge \\ & \neg(\exists e : \text{Entry} \mid e \in \text{ran } \text{entries} \bullet e.\text{name} = n?) \wedge \\ & \text{entries_1}' = \text{entries_1} \hat{\ } \\ & \quad \langle (\mu e : \text{Entry} \mid e.\text{name} = n? \wedge e.\text{phone} = \text{ph?}) \rangle \wedge \\ & \text{dom } \text{entries}' = \{e : \text{Entry} \mid e \in \text{ran } \text{entries_1}' \bullet e.\text{name}\} \wedge \\ & (\forall i \in \{1, \dots, \#\text{entries_1}'\} \bullet \\ & \quad \text{entries_1}'(i).\text{phone} = \text{entries}'(\text{entries_1}'(i).\text{name})) \Rightarrow \\ & (n? \notin \text{dom } \text{entries} \wedge \text{entries}' = \text{entries} \cup \{n? \mapsto \text{ph?}\}) \end{aligned}$$

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

Stärkere Nachbedingung (3)

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

Wir betrachten das Prädikat

$$\text{dom } \text{entries}' = \{e : \text{Entry} \mid e \in \text{ran } \text{entries_1}' \bullet e.\text{name}\}$$

Hier ist $\text{entries_1}'$

$$\begin{aligned} \text{entries_1}' &= \text{entries_1} \wedge \\ &\langle (\mu e_1 : \text{Entry} \mid e_1.\text{name} = n? \wedge e_1.\text{phone} = ph?) \rangle \end{aligned}$$

Folgerung: Es gibt in $\text{entries_1}'$ einen Eintrag e_1 mit Namen $n?$.

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

Stärkere Nachbedingung (4)

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

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

$$(\forall i \in \{1, \dots, \#\text{entries_1}'\} \bullet \text{entries_1}'(i).\text{phone} = \text{entries}'(\text{entries_1}'(i).\text{name}))$$

schließen wir, dass (A)

$$\begin{aligned} \text{entries}'(n?) &= \{i \in \{1, \dots, \#\text{entries_1}'\} \mid \\ &\text{entries_1}'(i).\text{name} = n? \bullet \text{entries_1}'(i).\text{phone}\} \end{aligned}$$

Aus dem Prädikat

$$\begin{aligned} \text{entries_1}' &= \text{entries_1} \wedge \\ &\langle (\mu e_1 : \text{Entry} \mid e_1.\text{name} = n? \wedge e_1.\text{phone} = ph?) \rangle \end{aligned}$$

wissen wir, dass (B)

$$\exists i \in \{1, \dots, \#\text{entries_1}'\} \bullet e.\text{name} = n? \wedge e.\text{phone} = ph?$$

Das Ziel (2) $\text{entries}'(n?) = ph?$ 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*

$$\frac{\text{iroot} : \mathbb{N} \rightarrow \mathbb{N}}{\forall a : \mathbb{N} \bullet (\text{let } r == \text{iroot}(a) \bullet \\ a \geq 0 \wedge \\ 0 \leq r * r \leq a < (r + 1) * (r + 1))}$$

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:

$$\frac{\{I\} \text{while}(p) \{p \wedge I\} S \{I' \wedge v' < v\}}{\{ \neg p \wedge I \}}$$

p : Wächter

v : Variante (ganze Zahl)

I : Invariante

Axiomatische Methode (3)

Wir instanziierten die axiomatische Definition der *while*-Anweisung:

```

{a ≥ 0}
r = 0;
{I}
while (p) {p ∧ I} r = r + 1; {I' ∧ v' < v}
{¬p ∧ I}
{0 ≤ r * r ≤ 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 ≥ 0}
r = 0;
{0 ≤ r * r ≤ a}
while (a ≥ (r + 1) * (r + 1)) r = r + 1; {v' < v}
{a < (r + 1) * (r + 1) ∧ 0 ≤ r * r ≤ a}
{0 ≤ r * r ≤ 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*

Spezifikations-Ausdruck $\sqsubseteq$ Ausdruck der Programmiersprache

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

Einfache Beispiele – Z nach C:

$$\begin{array}{ll} false & \sqsubseteq 0 & [C \text{ false}] \\ (p \wedge q) \vee (\neg p \wedge r) & \sqsubseteq p ? q : r & [\text{Bedingtes Prädikat}] \end{array}$$

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

$p \wedge q$	[gegeben]
$\Leftrightarrow (p \wedge q) \vee \text{false}$	$[p \vee \text{false} \Leftrightarrow p]$
$\Leftrightarrow (p \wedge q) \vee (\neg p \wedge \text{false})$	$[p \wedge \text{false} \Leftrightarrow \text{false}]$
$\sqsubseteq p ? q : \text{false}$	[Bedingtes Prädikat]
$\sqsubseteq p ? q : 0$	[C false]

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

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

Zustands-Schemata können zu *Typen* verfeinert werden:

Entry
name : NAME
phone : PHONE

wird in C zu

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

Zuweisung verfeinern

Die einfachste Verfeinerung ist die *unveränderte Variable*:

$$x' = x \sqsubseteq \langle \text{leere Anweisung} \rangle$$

Ansonsten eher trivial:

$$x' = e \sqsubseteq x' = e$$

Mehrfache Zuweisung kann aber haariger werden:

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

Spezielle Regeln für *komplexere Datenstrukturen* – etwa:

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

Bedingungen

Typisch: *Bedingter Zustandswechsel*

$$p \wedge s \sqsubseteq \text{if } (p) \ s$$

Disjunktion wird zu *Fallunterscheidung*:

$$(p \wedge s) \vee (q \wedge t) \sqsubseteq \text{if } (p) \ s; \text{ else if } (q) \ t$$

– setzt voraus, dass p und q disjunkt sind!

Konjunktionen

Konjunktionen können in der Regel nicht unmittelbar übersetzt werden.

Beispiel – Division:

$$\begin{aligned} \text{Quotient} &\hat{=} [n, d, q, r : \mathbb{Z} \mid d \neq 0 \wedge n = q * d + r] \\ \text{Remainder} &\hat{=} [r, d : \mathbb{Z} \mid r < d] \end{aligned}$$

Mögliche Verfeinerungen:

$$\begin{aligned} d \neq 0 \wedge n = q * d + r &\sqsubseteq q = 0; r = n \\ r < d &\sqsubseteq r = 0; \end{aligned}$$

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

Nein – es geht nur konstruktiv: etwa

for ($q = 0, r = n; r \geq d; q++$) $r = r - d;$

Quantoren

Quantoren sind leicht zu realisieren:

$$\begin{aligned}\forall x:S \bullet p(x) &\sqsubseteq b = 1; \text{ for}(x \in S) \text{ if}(!p(x)) b = 0; \\ \exists x:S \bullet p(x) &\sqsubseteq b = 0; \text{ for}(x \in S) \text{ if}(!p(x)) b = 1;\end{aligned}$$

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

$$\begin{aligned}S &\hat{=} [x, y : \mathbb{Z}] \\ Op &\hat{=} [\Delta S \mid x' = x + y;]\end{aligned}$$

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

$$Main \hat{=} Op_1 \vee Op_2 \vee \dots \vee Op_n \vee Exception$$

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