Programmbeweise mit Z

Andreas Zeller
Lehrstuhl für Softwaretechnik
Universität des Saarlandes, Saarbrücken

2005-11-28

Übersicht

- Das Schema-Kalkül
- Fallstudie: Textverarbeitung
- Formales Schließen
- Fallstudie: Expertensystem

Das Schema-Kalkül

Z besteht aus zwei Sprachen: der Sprache der gewöhnlichen Mathematik und der *Schema-Sprache.*

Schemata sind *das* charakteristische Merkmal von Z!

Sie ermöglichen

- Makro-ähnliches Einschließen und Wiederverwenden von Definitionen
- inkrementelle Spezifikation
- Spezifikation als Mischung aus Text und formalen Definitionen

Zustands-Schemata einschließen

Editor

$left, right : TEXT$

$\#(left \frown right) \leq maxsize$

Init

Editor ?

$left = right = \langle\,\rangle$

bedeutet tatsächlich

Init

$left, right : TEXT$

$\#(left \frown right) \leq maxsize$
$left = right = \langle\,\rangle$

Operations-Schemata einschließen

Insert

$\Delta Editor$?
$ch? : CHAR$

$ch? \in printing$
$left' = left \frown \langle ch? \rangle$
$right' = right$

Was bedeutet hier $\Delta Editor$?

Operations-Schemata einschließen (2)

Wir können Schema-Namen mit weiteren Zeichen (wie $'$) versehen. Dies wirkt sich auf alle im Schema definierten Variablen aus:

Editor

$left, right : TEXT$

$\#(left \frown right) \leq maxsize$

Editor′

$left', right' : TEXT$

$\#(left' \frown right') \leq maxsize$

Operations-Schemata einschließen (3)

ΔS ist für alle Schemata definiert als:

$$\begin{array}{|l}
\hline
\Delta S \\
S \\
S' \\
\hline
\end{array}$$

Expandieren wir $\Delta Editor$, so erhalten wir

$$\begin{array}{|l}
\hline
\Delta Editor \\
left, right : TEXT \\
left', right' : TEXT \\
\hline
\#(left \frown right) \leq maxsize \\
\#(left' \frown right') \leq maxsize \\
\hline
\end{array}$$

Operations-Schemata einschließen (4)

Somit expandiert das *Insert*-Schema zu

$$\begin{array}{|l}
\hline
Insert \\
left, right, left', right' : TEXT \\
ch? : CHAR \\
\hline
ch? \in printing \\
\#(left \frown right) \leq maxsize \\
\#(left' \frown right') \leq maxsize \\
left' = left \frown \langle ch? \rangle \\
right' = right \\
\hline
\end{array}$$

Operations-Schemata einschließen (5)

Ξ ist eine weitere Abkürzung – der Name des Schemata, bei dem sich nichts verändert:

$$
\begin{array}{|l}
\hline
\Xi Editor \\
\Delta Editor \\
\hline
left' = left \\
right' = right \\
\hline
\end{array}
$$

Expandiert:

$$
\begin{array}{|l}
\hline
\Xi Editor \\
left, right, left', right' : TEXT \\
\hline
\#(left \frown right) \leq maxsize \\
\#(left' \frown right') \leq maxsize \\
left' = left \\
right' = right \\
\hline
\end{array}
$$

Schema-Operatoren

$$
\begin{array}{|l}
\hline
Quotient \\
n, d, q, r : \mathbb{N} \\
\hline
d \neq 0 \\
n = q * d + r \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
Remainder \\
r, d : \mathbb{N} \\
\hline
r < d \\
\hline
\end{array}
$$

Was ist

$$Division \mathrel{\widehat{=}} Quotient \wedge Remainder \quad ?$$

Schema-Operatoren: Konjunktion

Quotient

$$n, d, q, r : \mathbb{N}$$

$$d \neq 0$$
$$n = q * d + r$$

Remainder

$$r, d : \mathbb{N}$$

$$r < d$$

Division ($\hat{=}$ *Quotient* $\land$ *Remainder*)

$$n, d, q, r : \mathbb{N}$$

$$d \neq 0$$
$$n = q * d + r$$
$$r < d$$

Schema-Operatoren: Übersicht

Bei der Schema-Konjunktion werden

- alle Deklarationen vereinigt
- alle Prädikate in einer *Konjunktion* zusammengefasst.

Analog – *Disjunktion*: Hier werden

- alle Deklarationen vereinigt
- alle Prädikate in einer *Disjunktion* zusammengefasst.

Schema-Operatoren: Disjunktion

DivideByZero

$$d, q, r : \mathbb{N}$$

$$d = 0 \land q = 0 \land r = 0$$

$$T\_Division \mathrel{\widehat{=}} Division \lor DivideByZero$$

ergibt

T_Division

$$n, d, q, r : \mathbb{N}$$

$$(d \neq 0 \land r < d \land n = q * d + r) \lor$$
$$(d = 0 \land r = 0 \land q = 0)$$

Schema-Operatoren: Negation

Die Negation eines Schemata ist nicht sehr nützlich.
Wir betrachten das expandierte EOF-Schema:

EOF

$$left, right : TEXT$$

$$\#(left \frown right) \leq maxsize$$
$$right = \langle\,\rangle$$

$\neg\, EOF$ ergibt

$\neg\, EOF$

$$left, right : TEXT$$

$$left \notin \operatorname{seq} CHAR \lor$$
$$right \notin \operatorname{seq} CHAR \lor$$
$$\#(left \frown right) > maxsize \lor$$
$$right \neq \langle\,\rangle$$

Schema-Operatoren: Negation (2)

Besser: Ein Schema, das gezielt Prädikate negiert

Aus

EOF

$$Editor$$

$$right = \langle\,\rangle$$

wird so

NotEOF

$$Editor$$

$$right \neq \langle\,\rangle$$

Fallstudie: Textverarbeitung

Aufgabe: Text in Wörter aufteilen

$$words\langle H, o, w, \ , a, r, e, \ , y, o, u\rangle = \langle\langle H, o, w\rangle, \langle a, r, e\rangle, \langle y, o, u\rangle\rangle$$

Wir definieren die Grundtypen:

$$[CHAR]$$

$$blank : \mathbb{P}\, CHAR$$

$$TEXT == \mathrm{seq}\, CHAR$$
$$SPACE == \mathrm{seq}_1\, blank$$
$$WORD == \mathrm{seq}_1 (CHAR \setminus blank)$$

Fallstudie: Textverarbeitung (2)

So sieht die Definition von *words* aus:

$$words : TEXT \to \mathrm{seq}\, WORD$$

$$\begin{aligned}
&\forall\, s : SPACE;\ w : WORD;\ l, r : TEXT \bullet \\
&\quad words(\langle\,\rangle) = \langle\,\rangle \land \\
&\quad words(s) = \langle\,\rangle \land \\
&\quad words(w) = \langle w\rangle \land \\
&\quad words(s \frown r) = words(r) \land \\
&\quad words(l \frown s) = words(l) \land \\
&\quad words(l \frown s \frown r) = words(l) \frown words(r)
\end{aligned}$$

Fallstudie: Textverarbeitung (3)

Wir möchten gerne die Anzahl der Zeilen und Wörter zählen – analog zum wc-Werkzeug in UNIX:

```
$ wc zed.tex
1060    2750   22413 zed.tex
$ _
```

Das ist nun nicht sehr schwierig:

$$wc : TEXT \to \mathbb{N} \times \mathbb{N} \times \mathbb{N}$$

$$\forall\, file : TEXT \bullet wc(file) = (\#lines(file), \#words(file), \#file)$$

Übung: Definieren Sie *lines*!

Fallstudie: Textverarbeitung (4)

Noch eine Aufgabe – Absätze auffüllen:

Aus

```
Viele Werkzeuge zur Versionskontrolle arbeiten
mit Sperren:
Zu jeder Zeit darf nur eine Person
das Dokument bearbeiten.
```

soll werden:

```
Viele Werkzeuge zur Versionskontrolle arbeiten mit Sperren:
Zu jeder Zeit darf nur eine Person das Dokument bearbeiten.
```

Fallstudie: Textverarbeitung (5)

$$width : \mathbb{N}$$

Format

$$t, t' : TEXT$$

$$words(t') = words(t)$$
$$\forall\, l : \operatorname{ran} lines(t') \bullet \#l \leq width$$

Fill

$$Format$$

$$\#lines(t') = min\{t' : TEXT \mid Format \bullet \#lines(t')\}$$

Übung: Was bedeutet dieses letzte Schema?

Formales Schließen

Beispiel:

Ein Zug bewegt sich mit einer konstanten Geschwindigkeit von 60 km/h. Wie weit kommt der Zug in 4 Stunden?

Als Z-Spezifikation:

$$\begin{array}{|l} distance, velocity, time : \mathbb{N} \\ \hline distance = velocity * time \\ velocity = 60 \\ time = 4 \end{array}$$

Frage: Was ist *distance*?

Beweis

Wir beweisen, dass $distance = 240$ gilt:

$$\begin{array}{lr} distance = velocity * time & \text{[Definition]} \\ \quad = 60 * time & [velocity = 60] \\ \quad = 60 * 4 & [time = 4] \\ \quad = 240 & \text{[Arithmetik]} \end{array}$$

Dies ist ein *strenges Vorgehen:* jeder Schritt wird durch eine Regel begründet.

Geht in Z auch kürzer:

$distance = velocity * time = 60 * time = 60 * 4 = 240$

Nicht-arithmetisches Schließen

$$\begin{array}{|l} philip : PERSON \\ adhesives, materials, research, manufactoring : \mathbb{P}\, PERSON \\ \hline adhesives \subseteq materials \\ materials \subseteq research \\ philip \in adhesives \end{array}$$

Frage: Gilt $philip \in research$?

$$\begin{array}{lr} \text{Ja, denn } philip \in adhesives & \text{[Definition]} \\ \quad \subseteq materials & \text{[Definition]} \\ \quad \subseteq research & \text{[Definition]} \end{array}$$

wobei wir aus $S \subseteq T \subseteq U \Rightarrow S \subseteq U$ schließen.

Noch ein Beweis

$$x : \mathbb{Z}$$
$$2 * x + 7 = 13$$

Wir schließen auf den Wert von x:

$2 * x + 7 = 13$	[Definition]
$\Leftrightarrow 2 * x = 13 - 7$	[Subtrahiere 7 auf beiden Seiten]
$\Leftrightarrow 2 * x = 6$	[Arithmetik]
$\Rightarrow (2 * x) \operatorname{div} 2 = 6 \operatorname{div} 2$	[Teile beide Seiten durch 2]
$\Leftrightarrow x = 6 \operatorname{div} 2$	[Division links; Algebra]
$\Leftrightarrow x = 3$	[Arithmetik]

Spezifikationen prüfen

Inkonsistenz: Spezifikationen, die sich gegenseitig widersprechen

Beispiel: *Existenz des Initial-Zustands*

$$\exists\, Editor \bullet Init$$

„Es gibt einen Editor, der das *Init*-Prädikat erfüllt“

Expandiert:

$$\exists\, left, right : TEXT \mid \#(left \frown right) \leq maxsize \bullet left = right = \langle\,\rangle$$

Offensichtlich wahr – aber wie beweist man so etwas formal?

Vorbedingungen berechnen

Die meisten Programme schlagen fehl, weil Programmierer Vorbedingungen nicht beachten:

Insert

$$\Delta Editor$$
$$ch? : CHAR$$

$$ch? \in printing$$
$$left' = left \frown \langle ch? \rangle$$
$$right' = right$$

Gibt es hier irgendwelche nicht-offensichtlichen Vorbedingungen?

Vorbedingungen berechnen (2)

Eine Vorbedingung beschreibt alle Zustände, in denen die Operation definiert ist.

Allgemeine Vorbedingung einer Operation Op auf einem Zustandsschema S ist, dass es ein *Folgeschema* S' gibt:

$$\exists S' \bullet Op$$

In unserem Fall:

$$\exists Editor' \bullet Insert$$

Oder kurz: Auch nach *Insert* gibt es immer noch einen *Editor*.

Vorbedingungen berechnen (3)

Editor

$$left, right : TEXT$$

$$\#(left \frown right) \leq maxsize$$

Aus

$$\exists Editor' \bullet Insert$$

wird

$$\exists left', right' : TEXT \mid \#(left' \frown right') \leq maxsize \bullet$$
$$ch? \in printing \land left' = left \frown \langle ch? \rangle \land right' = right$$

Vorbedingungen berechnen (4)

Wir haben:

$$\exists left', right' : TEXT \mid \#(left' \frown right') \leq maxsize \bullet$$
$$ch? \in printing \land left' = left \frown \langle ch? \rangle \land right' = right$$

Wir führen die Bedingungen des Existenzquantors in den Gültigkeitsbereich.

Generelle Regel – $(\exists d \mid p \bullet q) \Leftrightarrow (\exists d \bullet p \land q)$:

$$\exists left', right' : TEXT \bullet$$
$$ch? \in printing \land \#(left' \frown right') \leq maxsize \land$$
$$left' = left \frown \langle ch? \rangle \land right' = right$$

Vorbedingungen berechnen (5)

Wir haben:

$$\exists\, left', right' : TEXT \bullet$$
$$\quad ch? \in printing \land \#(left' \frown right') \leq maxsize \land$$
$$left' = left \frown \langle ch? \rangle \land right' = right$$

Wir entfernen den Existenzquantor mit der *Ein-Punkt-Regel*

$$(\exists\, x : T \bullet x = e \land p) \Leftrightarrow p[e/x]$$

($p[e/x]$: Ersetzen von e durch x in p)
und erhalten

$$ch? \in printing \land \#(left \frown \langle ch? \rangle \frown right) \leq maxsize$$

wobei wir $left' = left \frown \langle ch? \rangle$ und $right' = right$ angewandt haben.

Vorbedingungen berechnen (6)

Alles zusammen ($pr \equiv ch? \in printing$):

$\exists\, Editor' \bullet Insert$	[Definition der Vorbedingung]
$\Leftrightarrow left', right' : TEXT \mid \ldots \bullet \ldots$	[Schemata expandieren]
$\Leftrightarrow left', right' : TEXT \bullet \ldots \land \ldots$	[Eingeschränktes $\exists$]
$\Leftrightarrow pr \land \#(left \frown \langle ch? \rangle \frown right) \leq maxsize$	[Ein-Punkt-Regel]
$\Leftrightarrow pr \land \#left + \#\langle ch? \rangle + \#right \leq maxsize$	[$\#(s \frown t) = \#s + \#t$]
$\Leftrightarrow pr \land \#left + 1 + \#right \leq maxsize$	[$\#\langle x \rangle = 1$]
$\Leftrightarrow pr \land \#left + \#right < maxsize$	[Arithmetik]

Komplette Vorbedingung:

$$ch? \in printing \land \#left + \#right < maxsize$$

Fallstudie: Expertensystem

Fakten und Regeln:

$$[FACT]$$

$$stripes, fur, zebra, sharp\_teeth, carnivore, herbivore, mammal, tiger : FACT$$

$$rules : \mathbb{P}\, FACT \leftrightarrow FACT$$

$$\begin{aligned} rules = \{ & \\ & \vdots \\ & \{fur\} \mapsto mammal, \\ & \{sharp\_teeth\} \mapsto carnivore, \\ & \{stripes, mammal, carnivore\} \mapsto tiger, \\ & \{stripes, mammal, herbivore\} \mapsto zebra, \\ & \vdots \\ & \} \end{aligned}$$

Fallstudie: Expertensystem (2)

Monotones Schließen – Fakten werden zur Faktenbasis hinzugefügt

$$deduce == (\lambda\, facts : \mathbb{P}\, FACT \bullet facts \cup rules(\!|\, \mathbb{P}\, facts \,|\!))$$

Beispiel:

$$\begin{aligned} & facts = \{stripes, sharp\_teeth, fur\} \\ & deduce(facts) = \{stripes, sharp\_teeth, fur, mammal, carnivore\} \\ & deduce(deduce(facts)) = \{stripes, sharp\_teeth, fur, mammal, \\ & \quad carnivore, tiger\} \end{aligned}$$

Fallstudie: Expertensystem (3)

Transitiver Abschluss ($^+$) – Funktion wird angewandt, bis Fixpunkt erreicht

$$complete == deduce^+$$

so dass

$$\begin{aligned} & complete(facts) \\ & \quad = deduce(\ldots deduce(facts) \ldots) \\ & \quad = \{stripes, sharp\_teeth, fur, mammal, \\ & \quad carnivore, tiger\} \end{aligned}$$

Fallstudie: Expertensystem (4)

Wir sorgen für konsistente Regeln, indem wir *sich ausschließende Fakten* definieren:

$$
\begin{array}{l}
inconsistent == \{ \\
\quad \vdots \\
\quad \{fur, feathers, scales, \ldots\} \\
\quad \{mammal, bird, fish, \ldots\} \\
\quad \{tiger, zebra, ostrich, goldfish, \ldots\} \\
\quad \vdots \\
\}
\end{array}
$$

Fallstudie: Expertensystem (5)

Konsistente Fakten enthalten maximal ein Element jeder Menge:

$$
\begin{array}{|l}
\hline
consistent : \mathbb{P}(\mathbb{P}\, FACT) \\
\hline
\forall\, facts : consistent;\ mutually\_exclusive : inconsistent \bullet \\
\quad \#(mutually\_exclusive \cap facts) \leq 1 \\
\hline
\end{array}
$$

Hiermit schränken wir *complete* ein:

$$\forall\, facts : consistent \bullet complete(facts) \in consistent$$

Anders: *Aus einer konsistenten Menge von Fakten können wir nur konsistente Fakten schließen.*

Fallstudie: Expertensystem (6)

Herzstück des Schließens ist das *Deduce*-Schema:

Deduce

$$
\begin{array}{|l}
\hline
facts, facts' : \mathbb{P}\, FACT \\
data, goals, conclusions! : \mathbb{P}\, FACT \\
\hline
(\textbf{let}\ all == complete(facts \cup data) \bullet \\
\quad facts \subseteq facts' \subseteq all \land \\
\quad conclusions! = (goals \cap all) \subseteq facts') \\
\hline
\end{array}
$$

$facts$, $facts'$: Fakten (vorher und nachher)
$data$: Neue Fakten
$goals$: Menge der Ziele
$conclusions!$: Folgerungen (Ausgabe)

Fallstudie: Expertensystem (7)

Vorwärts-Schließen = Folgerungen aus Beobachtungen ableiten

Deduce

$facts, facts' : \mathbb{P}\, FACT$
$data, goals, conclusions! : \mathbb{P}\, FACT$

$(\textbf{let}\ all == complete(facts \cup data) \bullet$
$\quad facts \subseteq facts' \subseteq all \land$
$\quad conclusions! = (goals \cap all) \subseteq facts')$

Forward

$Deduce$
$observations? : \mathbb{P}\, FACT$

$data = observations?$

$observations? = \{stripes, sharp\_teeth, fur\} \land$
$goals = \{zebra, tiger, ostrich \ldots\} \Rightarrow conclusions! = \{tiger\}$

Fallstudie: Expertensystem (8)

Rückwärts-Schließen = Welche Anfragen passen zu den Fakten?

Deduce

$facts, facts' : \mathbb{P}\, FACT$
$data, goals, conclusions! : \mathbb{P}\, FACT$

$(\textbf{let}\ all == complete(facts \cup data) \bullet$
$\quad facts \subseteq facts' \subseteq all \land$
$\quad conclusions! = (goals \cap all) \subseteq facts')$

Backward

$Deduce$
$queries? : \mathbb{P}\, FACT$

$goals = queries?$

$facts = \{stripes, sharp\_teeth, fur\} \land data = \varnothing \land$
$queries? = \{tiger, zebra\} \Rightarrow conclusions! = \{tiger\}$

Fallstudie: Expertensystem (9)

Das *Deduce*-Schema suggeriert eine naive Implementierung:

1. *Alle* möglichen Fakten generieren (*all*)
2. Alle Fakten bestimmen, die auch in *goal* auftreten,
3. und als *Folgerungen* (*conclusions*!) zurückgeben

Eine tatsächliche Implementierung muss aber nicht so vorgehen (und sollte es auch nicht):

> *Die Spezifikation beschreibt nur die Wirkung, nicht das Vorgehen*

Alternative: *Ausführbare Spezifikation* (langsam, aber nützlich – für Prototypen oder Orakel)

Zusammenfassung

- Das Schema-Kalkül ermöglicht Einschließen, Konjunktion und Disjunktion von Schemata
- Schemata sind gute Grundlage für Programmbeweise
- Häufiger Programmbeweis: *Vorbedingungen berechnen*

Literatur

The Way of Z (Jonathan Jacky) – alle hier beschriebenen Beispiele und Tutorials

The Z Notation (http://www.comlab.ox.ac.uk/archive/z.html) – Die Z-Notation

The Z Glossary (ftp://ftp.comlab.ox.ac.uk/pub/Zforum/zglossary.ps.Z) – Z-Glossar

Fuzz Type Checker (http://spivey.oriel.ox.ac.uk/mike/fuzz/) – Typprüfer und LaTeX-Makros für Linux