Coalgebraische Methoden für Objektorientierte Spezifikation

Hendrik Tews

Institut für Theoretische Informatik, TU Dresden

http://wwwtcs.inf.tu-dresden.de/∼tews/

Coalgebren **leistungsstarke Theorembeweiser**

Spezifikation / Verifikation von Klassen objektorientierter Programmiersprachen

Kontext: LOOP — **L**ogic for **O**bject–**O**riented **P**rogramming

- Nijmegen – Dresden, seit ’97
- http://www.cs.kun.nl/∼bart/LOOP/

[1] H. Reichel. An approach to object semantics based on terminal co–algebras. *Mathematical Structure in Computer Science*, 5:129–152, 1995.

[2] U. Hensel, M. Huisman, B. Jacobs, and H. Tews. Reasoning about classes in object–oriented languages: Logical models and tools. In Ch. Hankin, editor, *European Symposium on Programming*, volume 1381 of *Lecture Notes in Computer Science*, pages 105–121. Springer, Berlin, 1998.

[3] H. Tews. Coalgebras for binary methods: Properties of bisimulations and invariants. *Theoretical informatics and applications*, 35(1):83–111, February 2001.

[4] J. Rothe, H. Tews, and B. Jacobs. The coalgebraic class specification language CCSL. *Journal of Universal Computer Science*, 7(2):175–193, March 2001.

[5] U. Hensel. *Definition and Proof Principles for Data and Processes*. PhD thesis, Univ. of Dresden, Germany, 1999.

[6] H. Tews. A case study in coalgebraic specification: Memory management in the FIASCO microkernel. Technical Report TPG2/1/2000, SFB 358, April 2000. Available at URL http://wwwtcs.inf.tu-dresden.de/∼tews/vfiasco/.

[7] D. Meyer. A case study in object oriented specification and verification: The MSMIE protocol. Graduation thesis, Catholic University of Nijmegen, January 1999.

[8] P. Lambooij. The YAPI protocol for buffered data transfer. Techn. Rep. CSI-R9923, Comput. Sci. Inst., Univ. of Nijmegen. Available at URL http://www.cs.kun.nl/csi/reports/info/CSI-R9923.html, 2000.

1. Generalisierte Coalgebren für beliebige polynomiale Signaturen —insbesondere für binäre Methoden— (CMCS '2000, [3])

2. CCSL — **C**oalgebraic **C**lass **S**pecification **L**anguage [4]
   - Spezifikationssprache basierend auf Algebren & Coalgebren
   - ausdrucksstarke coalgebraische Logik
   - coalgebraische Verfeinerung
   - Frontend für PVS und Isabelle/HOL (Isar)
   - Anwendbarkeit in Fallstudien bewiesen:
     - Speicherverwaltung im FIASCO Mikrokern [6]
     - MSMIE (multiprocessor shared-memory information exchange) Protokoll (Diplomarbeit Dijon Meyer, Nijmegen, [7])
     - YAPI (Y–chart Application Programmers Interface, Philips Research) Protokoll (Diplomarbeit Peter Lambooij, Nijmegen, [8])

3. Formalisierung von Resultaten & Beispielen in PVS

Inhalt

Definition 1 *Sei T ein Endofunktor auf* **Set** *— d.h. T ist eine Abbildung $X \longmapsto T(X)$. Eine T–***Coalgebra** *ist eine Funktion $\alpha : X \longrightarrow T(X)$.*

Beispiel

Seien A und B beliebige Mengen. Definiere

$$T_{\mathsf{Aut}}^{A,B}(S) \quad = \quad A \Rightarrow (S \times B) \quad = \quad \{f \mid f : A \longrightarrow S \times B\}$$

Eine T_{Aut}–Coalgebra $\alpha : S \longrightarrow A \Rightarrow S \times B$ korrespondiert zu zwei Funktionen

$$\begin{aligned} \mathsf{out}_\alpha \quad &: \quad S \times A \longrightarrow B \\ \mathsf{next}_\alpha \quad &: \quad S \times A \longrightarrow S \end{aligned}$$

$T_{\mathsf{Aut}}^{A,B}$–Coalgebren *sind* Automaten mit Eingabealphabet A und Ausgabealphabet B.

Theorie liefert (siehe Abschnitt 2.6, Seiten 50–71)

- Coalgebra–Homomorphismen (strukturverträgliche Abbildungen)
- Bisimulation/Bisimilarität (Verhaltensgleichheit)
- Invarianten
- viele Standardresultate, z.B.
  - Bisimulationen bilden einen vollständigen Verband
  - Coalgebra–Homomorphismen sind funktionale Bisimulationen

Beispiel Automatenhomomorphismem: Angenommen α und β sind Automaten.

$$\begin{array}{rcl} \mathsf{out}_\alpha & : & S \times A \longrightarrow B \\ \mathsf{next}_\alpha & : & S \times A \longrightarrow S \end{array} \qquad \begin{array}{rcl} \mathsf{out}_\beta & : & Q \times A \longrightarrow B \\ \mathsf{next}_\beta & : & Q \times A \longrightarrow Q \end{array}$$

Eine Funktion $f : S \longrightarrow Q$ ist ein Homomorphismus $\alpha \longrightarrow \beta$, falls für alle $s \in S$ und alle $a \in A$

$$\begin{array}{rcl} \mathsf{out}_\alpha(s, a) & = & \mathsf{out}_\beta(f(s), a) \\ f\big(\mathsf{next}_\alpha(s, a)\big) & = & \mathsf{next}_\beta(f(s), a) \end{array}$$

Definition 2 *Eine Coalgebra $\omega : Z \longrightarrow T(Z)$ ist* **final**, *falls es für jede Coalgebra $\alpha : X \longrightarrow T(X)$ genau einen Coalgebra Homomorphismus $!_\alpha : \alpha \longrightarrow \omega$ gibt.*

Verhaltenstypen Der Verhaltenstyp $\mathsf{Aut}[A, B]$ ist die finale $T^{A,B}_{\mathsf{Aut}}$–Coalgebra.

Theorem 3 *Die Bisimilarität für die finale Coalgebra ist in der Gleichheit enthalten.*

Coinduktion $=$ Ausnutzen der Finalität

Automatenverkettung Gegeben sind zwei Automaten α und β:

$$A \longrightarrow \boxed{\quad s \quad} \longrightarrow B \qquad\qquad B \longrightarrow \boxed{\quad q \quad} \longrightarrow C$$

Für jedes Alphabet A gibt es einen Automaten id_A mit einem Zustand und $\mathsf{out}_{\mathsf{id}_A}(s, a) = a$.

Proposition 4 *Für die Automatenverkettung gilt:* $\mathit{id}_A \circ \alpha \;=\; \alpha$

Beweis: Eine geeignete Bisimulation ist R mit

$$R(x, y) \quad \text{genau dann, wenn} \quad x = \mathsf{id}_A \circ y \qquad \square$$

```
Class 'a 'b Hash = {
      method find( name : 'a ) =                      /* nachschlagen */ ;
      method add( name : 'a, data : 'b ) =            /* hinzufügen */ ;
      method size =        ···     ;
}
```

Angenommen, X ist die Menge aller Objektzustände der Klasse Hash:

$$\begin{array}{lcl} \mathsf{find} & : & X \times {}'a \longrightarrow {}'b + \mathbf{1} \\ \mathsf{add} & : & X \times {}'a \times {}'b \longrightarrow X + X \\ \mathsf{size} & : & X \longrightarrow \mathbb{N} \end{array}$$

Ergibt eine *Coalgebra*

$$X \longrightarrow \big({}'a \Rightarrow {}'b + \mathbf{1}\big) \times \big({}'a \times {}'b \Rightarrow X + X\big) \times \mathbb{N}$$

(Zur Erinnerung: $X \times {}'a \longrightarrow {}'b + 1 \quad \cong \quad X \longrightarrow {}'a \Rightarrow {}'b + 1$)

<u>Class</u> A $=$ {

<u>method</u> equal(x : A) $=$ x $==$ <u>this</u> ;

<u>method</u> memory(env : $\mathbb{N} \Rightarrow$A) $=$ $\cdots$ env(5).equal(<u>this</u>) $\cdots$;

<u>method</u> strange(f : A $\Rightarrow$ A) $=$ $\cdots$;

}

Angenommen, X ist der Zustandsraum der Klasse A:

$$
\begin{array}{lcl}
\textsf{equal} & : & X \longrightarrow (X \Rightarrow \textsf{bool}) \\
\textsf{memory} & : & X \longrightarrow (\mathbb{N} \Rightarrow X) \Rightarrow \boxed{\;\cdots\;} \\
\textsf{strange} & : & X \longrightarrow (X \Rightarrow X) \Rightarrow \boxed{\;\cdots\;}
\end{array}
$$

Endofunktoren können binäre Methoden prinzipiell nicht modellieren!

(Binäre Methoden sind problematisch in allen Ansätzen zur theoretischen Fundierung objektorientierter Programmiersprachen)

Lösung: Benutze Funktoren $\mathbf{Set}^{op} \times \mathbf{Set} \longrightarrow \mathbf{Set}$!

Funktorklasse	Eigenschaft	Beispiel
polynomial	konstante Argumente	$\mathbf{Self} \times A \longrightarrow \boxed{\cdots}$
erweitert kartesisch	keine Funktionsargumente	$\mathbf{Self} \times \mathbf{Self} \times \mathbf{Self} \longrightarrow \boxed{\cdots}$ $\mathbf{Self} \times (\mathbf{Self} + A) \longrightarrow \boxed{\cdots}$
erweitert polynomial	funktionale Argumente mit konst. Def.-bereich	$\mathbf{Self} \times (A \Rightarrow \mathbf{Self}) \longrightarrow \boxed{\cdots}$
höhere Ordnung	beliebige Argumente	$\mathbf{Self} \times (\mathbf{Self} \Rightarrow A) \longrightarrow \boxed{\cdots}$ $\mathbf{Self} \times (\mathbf{Self} \Rightarrow \mathbf{Self}) \longrightarrow \boxed{\cdots}$

Sei $\mathbb{C}$ eine bikartesisch abgeschlossene Kategorie (d.h. mit Produkten, Summen und Exponenten).

polynomiale Funktoren: $F : \mathbb{C} \longrightarrow \mathbb{C}$ bzw. $H : \mathbb{C}^{op} \times \mathbb{C} \longrightarrow \mathbb{C}$

$$\begin{array}{rcl} F(X) & = & A \mid X \mid F_1(X) \times F_2(X) \mid F_1(X) + F_2(X) \\ H(Y,X) & = & A \mid X \mid H_1(Y,X) \times H_2(Y,X) \mid H_1(Y,X) + H_2(Y,X) \end{array}$$

	Exponent
kartesisch	(kein Exponent)
polynomial	$A \Rightarrow F_1(X)$
erweitert kartesisch	$K(Y) \Rightarrow H_1(Y,X)$
erweitert polynomial	$F(Y) \Rightarrow H_1(Y,X)$
höherer Ordnung	$H_1(X,Y) \Rightarrow H_2(Y,X)$

Coalgebren sind Morphismen $X \xrightarrow{c} H(X,X)$

Homomorphismen

$$\begin{tikzcd} X \arrow[r, "c"] \arrow[d, "f"'] & H(X,X) \arrow[dr, "{H(X,f)}"] & \\ & & H(X,Y) \\ Y \arrow[r, "d"] & H(Y,Y) \arrow[ur, "{H(f,Y)}"'] & \end{tikzcd}$$

Proposition 5 *Sei $\mathcal{A}$ eine nichttriviale ($H_{\mathcal{A}}(\mathbf{1},\mathbf{1}) \not\cong \mathbf{1}$) Klassensignatur mit einer binären Methode m. Dann gibt es keine finale Coalgebra für $\mathcal{A}$.*

Beweis

1. Angenommen $H_{\mathcal{A}}$ hat eine finale Coalgebra $z : Z \longrightarrow H(Z, Z)$.
2. Konstruiere eine neue Coalgebra $z' : Z + 1 \longrightarrow H(Z + 1, Z + 1)$ sodass
   - die Einbettung $\iota : Z \longrightarrow Z + 1$ ein Coalgebra–Homomorphismus ist *und*
   - für alle $y \in Z : m_{z'}(\perp, \iota\, y) \neq m_z(y, y)$.
3. Sei $f : z' \longrightarrow z$ ein Coalgebra Homomorphismus :
$$\begin{tikzcd} Z \arrow[loop left, "\mathsf{id}_Z"] \arrow[rr, bend left, "\iota"] & & Z + 1 \arrow[ll, bend left, "f"] \end{tikzcd}$$
4. $$\begin{aligned} m_z(f\perp, f\perp) &= m_z\big(f\perp, f(\iota(f\perp))\big) \\ &= m_{z'}(\perp, \iota(f\perp)) \\ &\neq m_z(f\perp, f\perp) \end{aligned}$$

□

Funktorklasse	Generalisierung Eigenschaften	
polynomial		finale Coalgebren/Coinduktion vollst. Verband von Bisimulationen + viel mehr
erweitert kartesisch		keine finalen Coalgebren/Coinduktion keine Vereinigung von Bisimulationen vollst. Verband von Bisimulations*äquivalenzen*
erweitert polynomial		Schnitt von Bisimulationen keine Vereinigung von Bisimulationen unvollst. Verband von Bisimulationen + die meisten Standardresultate
höhere Ordnung		nur triviale Eigenschaften z.B. Gleichheit ist Bisimulation kein Schnitt/Vereinigung von Bisimulationen

Für alle negativen Resultate gibt es Gegenbeispiele in PVS!

Sei G ein erweitert polynomialer Funktor und sei H ein polynomialer Funktor höherer Ordnung.

Proposition 6 *Falls R eine H–Bisimulation ist, dann auch die inverse Relation R^{op}.*

Proposition 7 *Relationen–Lifting und Prädikaten–Lifting für erweiterte polynomiale Funktoren ist gefasert (d.h. kommutiert mit Substitution).*

Proposition 8 *Eine Funktion ist ein G–Coalgebra Homomorphismus genau dann, wenn ihr Graph eine Bisimulation ist.*

Proposition 9 *Bisimulationen und Invarianten für erweiterte polynomiale Funktoren sind gegenüber Schnittbildung abgeschlossen.*

Proposition 10 *Die Relation $\pi_1^* P \cap R$ ist eine G–Bisimulation für alle G–Invarianten P und G–Bisimulationen R.*

CCSL = Coalgebren + HOL

Eine Klassenspezifikation $\mathcal{A}$ besteht aus

- Signatur
  - Methoden Deklarationen
  - Konstruktor Deklarationen
- Axiomen
  - Methodenaxiomen $\mathcal{A}_M$
  - Konstruktoraxiomen $\mathcal{A}_C$
- syntaktischem Zucker
  - Vererbungsklauseln (mit Umbenennung)
  - definitorische Erweiterungen
  - Theoreme
  - Importklauseln
  - Lifting Requests

Die Logik für die Axiome ist signaturabhängig.

- Logik höherer Ordnung (HOL)
- Verhaltensgleichheit
- (methodenweise) modale Operatoren $\Box$ (always) und $\Diamond$ (eventually)

```
Begin Queue[ A : Type ] : ClassSpec
    Method
      put : [Self, A] -> Self;
      top : Self -> Lift[[A, Self]];        % = {⊥} ∪ {up(a, x) | a ∈ A, x ∈ Self}

    Constructor
        new : Self;

    Assertion
      SelfVar x: Self

      q_empty :
        x.top ∼ ⊥ Implies
          Forall(a : A) . x.put(a).top ∼ up(a,x);

      q_filled :
        Forall(a1 : A, y : Self) . x.top ∼ up(a1, y) Implies
          Forall(a2 : A) . x.put(a2).top ∼ up(a1, y.put(a2));

    Creation
      q_new : new.top ∼ ⊥;
END Queue
```

```
Begin SetSig[X : Pos Type] : GroundSignature
  Importing Setops[X]
  Type setof
  Constant
    emptyset : setof[X];
    member : [[X, setof[X]] -> bool];
    pred2set : [[X -> bool] -> setof[X]];

    Everysetof : [[X -> bool] -> [setof[X] -> bool]]
      Everysetof P xs = Forall(x : X) : member(x,xs) IMPLIES P x;

    RelEverysetof[Y : Type] : [[X,Y -> bool] -> [setof[X],setof[Y] -> bool]]
    RelEverysetof(R)(xs,ys) =
      ((Forall(x : X) : member(x,xs) Implies
          Exists(y : Y) : member(y,ys) And R(x,y))
        AND
       (Forall(y : Y) : member(y,ys) Implies
          Exists(x : X) : member(x,xs) And R(x,y)));

    Mapsetof[Y : Type] : [[X -> Y] -> [setof[X] -> setof[Y]]]
    Mapsetof f xs = pred2set(Lambda(y : Y) :
      Exists(x : X) : member(x,xs) And y = f x);
End SetSig
```

siehe Abschnitte 4.1–4.7, Seiten 125–207

Modelle

Sei $\mathcal{A}$ eine Klassenspezifikation:

Signatur:

Methodendeklarationen $\Longrightarrow$ Methodenfunktor $H_{\mathcal{A}}$.

Konstruktordeklarationen $\Longrightarrow$ Konstruktorfunktor $F_{\mathcal{A}}$.

Modell für Signatur: ist eine Menge S mit zwei Funktionen

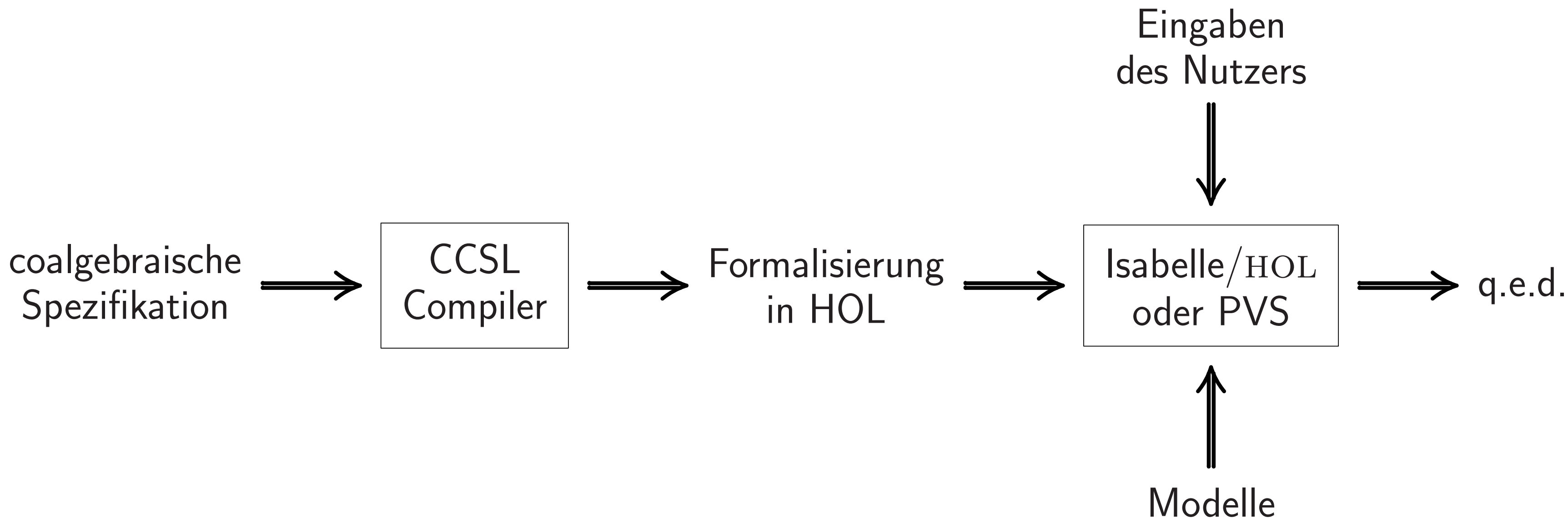
$$F_{\mathcal{A}}(S) \xrightarrow{\quad a \quad} S \xrightarrow{\quad c \quad} H_{\mathcal{A}}(S, S)$$

mit denen sich alle Operationen der Spezifikation $\mathcal{A}$ interpretieren lassen.

Interpretation: Die Methodenaxiome ergeben ein Prädikat $P \subseteq S$ und die Konstruktoraxiome ein Prädikat $Q \subseteq F(S) \Rightarrow S$.

Modell: Ein Tupel (S, c, a) ist ein Model für $\mathcal{A}$, wenn

$$\forall s \in S \,.\, P(s) \qquad \text{und} \qquad a \in Q$$



Implementation

- in Ocaml implementiert
- einfachste Compilertechnik, > 10 Pässe
- 40.000 Zeilen, 100 Dateien
- Version 2.2
- frei verfügbar unter http://wwwtcs.inf.tu-dresden.de/~tews/ccsl

```
QueueInterface[Self : TYPE , A : TYPE] : THEORY
BEGIN
  IMPORTING Lift[[A , Self]]

  % Declare coalgebraic signature for class Queue

  QueueSignature : TYPE =
    [#
      put : [[Self , A] -> Self] ,
      top : [Self -> Lift[[A , Self]]]
    #]

  % Declare constructor signature for class Queue

  QueueConstructors : TYPE = [# new : Self #]

END QueueInterface
```

[1] M. Abadi and L. Cardelli. *A Theory of Objects*. Springer-Verlag, New York, 1996.

[2] C. Cîrstea. A coalgebraic equational approach to specifying observational structures. In B. Jacobs and J. Rutten, editors, *Coalgebraic Methods in Computer Science '99*, volume 19 of *ENTCS*. Elsevier, Amsterdam, 1999.

[3] R. Hennicker and A. Kurz. (Ω, Ξ)–Logic: On the algebraic extension of coalgebraic specifications. In B. Jacobs and J. Rutten, editors, *Coalgebraic Methods in Computer Science '99*, volume 19 of *ENTCS*, pages 195–212. Elsevier, 1999.

[4] P. D. Mosses. CoFI: The common framework initiative for algebraic specification and development. In M. Bidoit and M. Dauchet, editors, *TAPSOFT '97: Theory and Practice of Software Development*, volume 1214 of *Lecture Notes in Computer Science*, pages 115–140. Springer-Verlag, 1997.

[5] T. Mossakowski. CASL: From semantics to tools. In S. Graf and M. Schwartzbach, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1785 of *Lecture Notes in Computer Science*, pages 93–108. Springer, 2000.

[6] Grigore Roşu. *Hidden Logic*. PhD thesis, University of California at San Diego, 2000.

[7] P. Kellomäki. *Mechanical Verification of Invariant Properties of DisCo Specifications*. PhD thesis, Tampere University of Technology, Finland, 1997.

[8] J. M. Spivey. *The Z Notation: A Reference Manual*. Prentice-Hall, 2nd edition, 1992.

[9] K. Lano. *The B Language and Method: A guide to Practical Formal Development*. Springer Verlag London Ltd., 1996.

Zusammenfassung

- rigorose Lösung für binäre Methoden
  verantwortungsvoller Nutzer wählt Kompromiss
  zwischen Ausdrucksstärke und strukturellen Eigenschaften
- Paradigma *Klassen sind Coalgebren* ist tragfähig
- CCSL/PVS ist eine leistungsstarke und erprobte Umgebung
  zur Spezifikation und Verifikation objektorientierter Programme

Ausblick

- Ideen für CCSL Version 3
- Erweiterung von CCSL um algebraische Spezifikationen
- effizientes Schließen mit Bisimulationen
- Logik mit allgemeinen Fixpunktoperatoren
- parallele Coalgebren
- Projekt VFiasco (http://www.vfiasco.de)