The Coalgebraic Class Specification Language CCSL

Hendrik Tews

Institut für Theoretische Informatik, TU Dresden
tews@tcs.inf.tu-dresden.de

Contents

Coalgebra is a function $X \longrightarrow \boxed{\cdots\ X\ \cdots}$

for instance $\mathsf{Aut}_{A,B} \xrightarrow{\langle out, next \rangle} A \Rightarrow (\mathsf{Aut}_{A,B} \times B)$

Morphisms of Coalgebras as you would expect:

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

Important concepts

- Bisimulation $x\,R\,y \iff x$ and y show the same behaviour
  defined via morphisms [AM89] or via relation lifting [HJ98]
- Invariant if $x \in P$ then all successors of x are also in P
- final coalgebra most abstract and contains all possible behaviours

Coalgebras can model Classes of object-oriented languages

class Point
  methods
    get_x : **Self** $\longrightarrow$ real
    get_y : **Self** $\longrightarrow$ real
    move : **Self** $\times$ real $\times$ real $\longrightarrow$ **Self**
end Point

Coalgebra $c : X \longrightarrow T_P(X)$ with

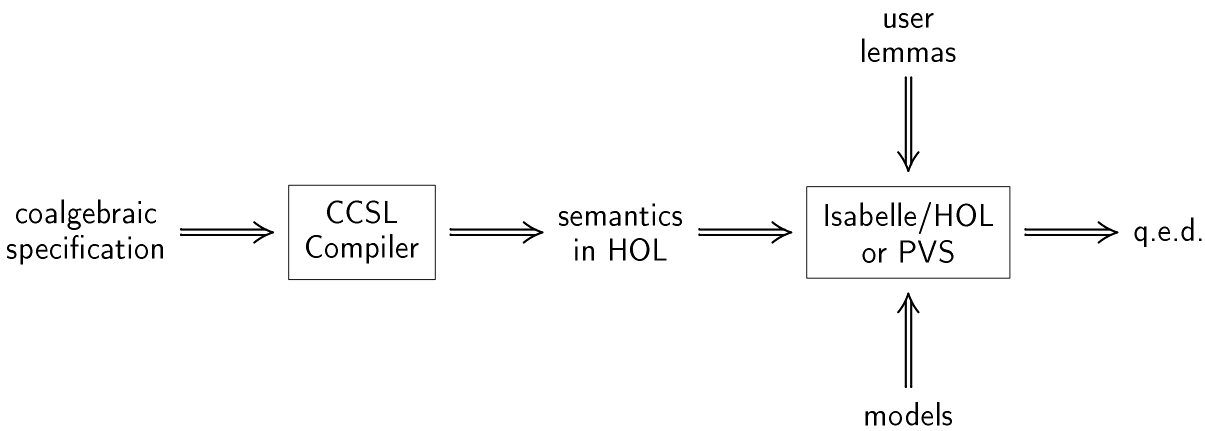
$$T(X) = \mathbb{R} \times \mathbb{R} \times (\mathbb{R} \times \mathbb{R} \Rightarrow X)$$

---

algebraic data types
$+$
coalgebraic class specifications
$+$
higher-order logic

$=$ **CCSL**

CCSL Compiler



```
Begin Queue[ A : Type ] : ClassSpec
  Method
    put : [Self, A] -> Self;
    top : Self -> Lift[[A,Self]];
  Constructor
    new : Self;
  Assertion Selfvar x : Self
    q_empty :
      x.top ~ bot 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 ~ bot;
  Theorem Importing QueueModal[Self,A]
    strong_reachable : Forall(p, q : Self) :
      Let finite? : [Self -> bool] = Lambda(q : Self) :
               (Eventually Lambda(x : Self) : x.top = bot For {top}) q IN
        finite? p And finite? q   Implies
          (Eventually (Eventually Lambda(r : Self) : r ~ q For {put})
           For {top}
          ) p;
End Queue
```

- parametrised class specifications
  - inheritance (of specifications)
  - methods, constructor declarations
  - binary methods
  - attributes & update operations
  - final or loose semantics
  - axioms for methods and constructors
  - behavioural equality
  - method-wise infinitary modal operators
  - theorems
- parametrised abstract data types
  - initial semantics
  - no axioms
- ground signature extentions
  - parametric Types & Terms
  - definitions & declarations
- ADT's and Classes can be nested $\Longrightarrow$ nested induction and coinduction
- higher-order logic
- support for PVS and Isabelle/HOL
- coalgebraic refinement [JT01]
- Version 2.2 available at http://wwwtcs.inf.tu-dresden.de/∼tews/ccsl

Invariant is a predicate closed under the coalgebra

in the Queue example: $q \in P \Longrightarrow \forall a \,.\, \mathsf{put}(q, a) \in P$

Invariants form a complete lattice

$\underline{\boldsymbol{P}}$ denotes the greatest invariant contained in P

$$\underline{P} \quad = \quad \bigvee \{Q \mid Q \subseteq P \ \wedge \ Q \text{ is an invariant}\}$$

Observe $x \in \underline{P}$ then P holds on all direct and indirect successor states of x

$\Box$ (always) acts on predicates (like a variable binder)

$$\Box : \big(\textsf{\textbf{Self}} \longrightarrow \textsf{Prop}\big) \longmapsto \big(\textsf{\textbf{Self}} \longrightarrow \textsf{Prop}\big)$$

in the logic: $\Gamma, x : \textsf{\textbf{Self}} \vdash \varphi$ gives $\Gamma, y : \textsf{\textbf{Self}} \vdash (\Box\lambda x : \textbf{Self} \,.\, \varphi)\,(y)$.

Semantics

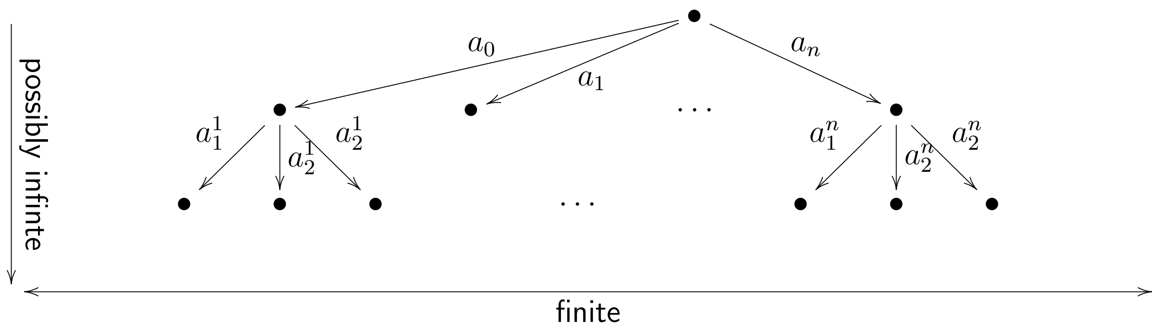
$$[\![\Box_x\,\varphi\,(y)]\!] \quad = \quad \big\{(\vec{a}, b) \in [\![\Gamma]\!] \times [\![\textsf{\textbf{Self}}]\!] \;\mid\; b \in \underline{\{x \in [\![\textsf{\textbf{Self}}]\!] \;\mid\; [\![\varphi]\!](\vec{a}, x)\}}\;\big\}$$

Eventually $\quad \Diamond\varphi \;\stackrel{\textsf{def}}{=}\; \neg\Box\neg\varphi \qquad \Big(\stackrel{\textsf{def}}{=}\; \lambda x : \textbf{Self} \,.\, \neg(\Box\lambda y : \textbf{Self} \,.\, \neg\varphi(y))\,(x)\Big)$

Method-wise modal operators arise from subcoalgebras

```
Begin list[ X : Type ] : Adt
  Constructor
    null : Carrier;
    cons( car, cdr ) : [X, Carrier] -> Carrier
End list

Begin InfTreeFin[A : Type] : Final ClassSpec
  Method
    next : Self -> list[[A,Self]];
End InfTreeFin
```



	Induction on lists	**Coinduction on InfTreeFin**
definition	Exposing a list algebra on a set X uniquely defines list$\longrightarrow X$	Exposing an InfTreeFin coalgebra structure on X uniquely defines $X \longrightarrow$InfTreeFin
proof	Any inductive predicate holds on all lists	Any InfTreeFin bisimulation is contained in the diagonal (i.e., $x\,R\,y \Longrightarrow x = y$)

Define empty_tree via $\mathsf{next}(\perp) = \mathsf{null}$

Define concat via $\mathsf{next}(p, q) = \begin{cases} \mathsf{map}(\lambda a, p' \,.\, (a, (p', q)))\,(\mathsf{next}\, p) & \text{if } \mathsf{next}\, p \neq \mathsf{null} \\ \mathsf{map}(\lambda a, q' \,.\, (a, (p, q')))\,(\mathsf{next}\, q) & \text{if } \mathsf{next}\, p = \mathsf{null} \end{cases}$

Prove $\mathsf{concat}(\mathsf{empty\_tree}, p) \;=\; p$ (do induction inside a coinduction)

Validity investigated in [Hen99]

In general

- "object orientation" applies to software construction systems
- it does *not* apply to specification languages
- CCSL is a specification language *for* object-oriented programs
- CCSL gives general support —
  it does not specialise to the peculiar semantics of a special OO language

---

CCSL supports

- multiple inheritance
- overriding of methods
- late binding possible
- object identity possible
- dynamic type checking
  (down casts) possible

Terminology

class specification

class is a model $\langle X, c : X \longrightarrow T(X) \rangle$

(state of an) object is an element $x \in X$

Inheritance of specifications

- inherits methods & method axioms
- renaming possible
- simple semantics: Let $\mathcal{B}$ inherit from $\mathcal{A}$, then

$$T_B(X) \quad = \quad T_A(X) \quad \times \quad \boxed{\text{new methods of } \mathcal{B}}$$

Signature model of $\mathcal{B}$: $\langle Y, d : Y \longrightarrow T_B(Y) \rangle$

Semantics of the method axioms wrt. d: $[\![\mathcal{M}_\mathcal{B}]\!]_d : Y \longrightarrow \mathsf{Prop}$

$$[\![\mathcal{M}_\mathcal{B}]\!]_d \quad = \quad [\![\mathcal{M}_\mathcal{A}]\!]_{\pi_1 \circ d} \quad \wedge \quad \boxed{\text{new axioms of } \mathcal{B}}$$

- π_1 is like an upcast — but is does not convert the type
- For final semantics: $!_{\pi_1 \circ d}$ is an static upcast that converts the type of the objects
- Inheritance on specifications does not require inheritance of models

Program

```
class parent = object
  method m = 0
end
```

```
class heir = object
  inherit parent
  method m = 1
end
```

Simple approach

```
Begin SParent : ClassSpec
  Method
    m : Self -> nat;
  Assertion Selfvar x : Self
    p1 : x.m <= 1;
End SParent
```

```
Begin SHeir : Classspec
  Inherit From SParent
  Assertion Selfvar x : Self
    h1 : x.m = 1;
End SHeir
```

More complex approach

```
Begin SParent : ClassSpec
  Method
    dynamic_type : Self -> nat;
    m : Self -> nat;
  Assertion Selfvar x : Self
    p1 : x.m <= 1;
    p2 : dynamic_type(x) = 0
           Implies x.m = 0;

  Constructor
    new_parent : Self;
  Creation
    p3 : dynamic_type(new_parent) = 0
End SParent
```

```
Begin SHeir : ClassSpec
  Inherit From SParent
  Assertion Selfvar x : Self
    h1 : dynamic_type(x) = 1
           Implies x.m = 1;

  Constructor
    new_heir : Self;
  Creation
    h2 : dynamic_type(new_heir) = 1
End SHeir
```

Binary Methods—Example

<u>Class</u> A = {

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

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

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

}

Assume, that X is the state space of the class

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

Endofunctors cannot describe such signatures!

- use bivariant functors $H : \mathbf{Set}^{op} \times \mathbf{Set} \longrightarrow \mathbf{Set}$
- Coalgebras are morphisms $X \xrightarrow{\ c\ } H(X, X)$
- Morphism of coalgebras:

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

- restrict to *polynomial functors*
  inductively built from constants, identity, $+$, $\times$, and $\Rightarrow$
- see [Tew01, Tew02b, Tew02a]

Class of Functors	Property	Example
cartesian $\quad \mathbf{Set} \longrightarrow \mathbf{Set}$ $K \;=\; A \mid \mathsf{Id} \mid K_1 \overset{+}{\times} K_2$	no arguments	$\textsf{Self} \longrightarrow \boxed{\cdots}$
polynomial $\quad \mathbf{Set} \longrightarrow \mathbf{Set}$ $F \;=\; A \mid \mathsf{Id} \mid F_1 \overset{+}{\times} F_2 \mid A \Rightarrow F_1$	constant arguments	$\textsf{Self} \times A \longrightarrow \boxed{\cdots}$
extended cartesian $\quad \mathbf{Set}^{op} \times \mathbf{Set} \longrightarrow \mathbf{Set}$ $G^K \;=\; A \mid \mathsf{Id} \mid G_1^K \overset{+}{\times} G_2^K \mid K \Rightarrow G_1^K$	no functional arguments	$\textsf{Self} \times \textsf{Self} \times \textsf{Self} \longrightarrow \boxed{\cdots}$ $\textsf{Self} \times (\textsf{Self} + A) \longrightarrow \boxed{\cdots}$
extended polynomial $\quad \mathbf{Set}^{op} \times \mathbf{Set} \longrightarrow \mathbf{Set}$ $G^F \;=\; A \mid \mathsf{Id} \mid G_1^F \overset{+}{\times} G_2^F \mid F \Rightarrow G_1^F$	functional arguments with constant domain	$\textsf{Self} \times (A \Rightarrow \textsf{Self}) \longrightarrow \boxed{\cdots}$
higher-order polynomial $\quad \mathbf{Set}^{op} \times \mathbf{Set} \longrightarrow \mathbf{Set}$ $H \;=\; A \mid \mathsf{Id} \mid H_1 \overset{+}{\times} H_2 \mid H_1 \Rightarrow H_2$	arbitrary arguments	$\textsf{Self} \times (\textsf{Self} \Rightarrow A) \longrightarrow \boxed{\cdots}$ $\textsf{Self} \times (\textsf{Self} \Rightarrow \textsf{Self}) \longrightarrow \boxed{\cdots}$

Consider a higher-order polynomial functor H and two coalgebras $c : X \longrightarrow H(X, X)$ and $d : Y \longrightarrow H(Y, Y)$.

Bisimulations according to Hermida & Jacobs [HJ98]

- R is a bisimulation for c and d, if

$$\forall x \in X, y \in Y \,.\, R(x, y) \quad \text{implies} \quad \mathrm{Rel}(H)(R, R)(c(x), d(y))$$

Bisimulations according to Aczel & Mendler [AM89]

$$\begin{tikzcd}
X \arrow[d, "c"'] & & R \arrow[ll, "\pi_1"'] \arrow[rr, "\pi_2"] \arrow[d] & & Y \arrow[d, "d"] \\
H(X,X) \arrow[dr, "H(\pi_1, X)"'] & & H(R,R) \arrow[dl, "H(R,\pi_1)"] \arrow[dr, "H(R,\pi_2)"'] & & H(Y,Y) \arrow[dl, "H(\pi_2, Y)"] \\
& H(R,X) & & H(R,Y) &
\end{tikzcd}$$

Both notions coincide for extended polynomial functors (in general they are different)

Relation lifting (for a h.-o. pol. functor H) is an operation on relations

$$\big(S \subseteq U \times V, \quad R \subseteq X \times Y\big) \longmapsto \mathsf{Rel}(H)(S,R) \quad \subseteq \quad H(U,X) \times H(V,Y)$$

Defined by induction on the structure of H:

H	$\mathrm{Rel}(H)$	$a \in H(U,X) \quad b \in H(V,Y)$
X	R	$R(a,b)$
A	$=_A$	$a =_A b$
$H_1 \times H_2$	$\mathrm{Rel}(H_1) \times_{\mathrm{R}} \mathrm{Rel}(H_2)$	$\mathsf{Rel}(H_1)(S,R)(\pi_1\, a, \pi_1\, b) \;\wedge\; \mathsf{Rel}(H_2)(S,R)(\pi_2\, a, \pi_2\, b)$
$H_1 + H_2$	$\mathrm{Rel}(H_1) +_{\mathrm{R}} \mathrm{Rel}(H_2)$	$\begin{cases} \text{if } (a,b) = (\kappa_1\, a_1, \kappa_1\, b_1) & \mathsf{Rel}(H_1)(S,R)(a_1,b_1) \\ \text{if } (a,b) = (\kappa_2\, a_2, \kappa_2\, b_2) & \mathsf{Rel}(H_2)(S,R)(a_2,b_2) \\ \text{otherwise} & \bot \end{cases}$
$H_1 \Rightarrow H_2$	$\mathrm{Rel}(H_1) \Rightarrow_{\mathrm{R}} \mathrm{Rel}(H_2)$	$\forall (u,v) \in \mathsf{Rel}(H_1)(R,S) \;.\; \mathsf{Rel}(H_2)(S,R)(a(u), b(v))$

Extension

polynomial	structural properties + greatest bisimulations + final models
extended cartesian	structural properties + greatest bisimulations equivalences for all coalgebras
extended polynomial	structural properties + greatest bisimulations equivalences for finitely based coalgebras
higher-order	only trivial properties

- Results/Examples have been checked/developed with PVS.
  see `http://wwwtcs.inf.tu-dresden.de/~tews/binary/`
- can allow list : $\mathbf{Set} \longrightarrow \mathbf{Set}$

Let $G : \mathbf{Sets}^{op} \times \mathbf{Sets} \longrightarrow \mathbf{Sets}$ be a extended polynomial functor.

Proposition 1 *Relation lifting for G is fibred, i.e. commutes with substitution.*

Proposition 2 *Bisimulations and invariants are closed under arbitrary nonempty intersections.*

Proposition 3 *Bisimulations are closed under composition.*

Proposition 4 *The image $\coprod_f P$ of a morphism with respect to an invariant P is an invariant.*

Proposition 5 *Invariants correspond to subcoalgebras and vice versa.*

Proposition 6 *The predicate $\coprod_{\pi_1} R$ is an invariant for any bisimulation R.*

Proposition 7 *The relation $\pi_1^* P \cap R$ is a bisimulation for any invariant P and any bisimulation R.*

Proposition 8 *The predicate P is an invariant if and only if $\coprod_\delta P$ is a bisimulation.*

Proposition 9 *A function is a G–coalgebra morphism if and only if its graph is a bisimulation.*

Proposition 10 *The kernel of a G–morphism is a bisimulation.*

Proposition 11 *The Aczel/Mendler approach and the Hermida/Jacobs approach yield identical notions of invariant and bisimulation.*

Proposition 12 *If a class signature $\mathcal{A}$ contains a binary method m, then the associated signature functor $H_{\mathcal{A}}$ has a final coalgebra if and only if $H_{\mathcal{A}}(\mathbf{1},\mathbf{1})$ is a singleton.*

Proof

1. Assume H has a final coalgebra, say $z : Z \longrightarrow H(Z, Z)$.
2. Construct a new coalgebra $z' : Z + 1 \longrightarrow H(Z+1, Z+1)$ such that
   - the inclusion $\iota : Z \longrightarrow Z + 1$ is a coalgebra morphism *and*
   - for all $y \in Z$: $m_{z'}(\perp, \iota\, y) \neq m_z(y, y)$.
3. Note, that if $f : Z + 1 \longrightarrow Z$ is a coalgebra morphism $z' \longrightarrow z$, then $f \circ \iota = \mathrm{id}_Z$.
4. Conclude, that there is no such morphism $z' \longrightarrow z$ by construction:

$$m_z(\circledast, \circledast) \quad = \quad m_z(\circledast, (f \circ \iota)\circledast) \quad = \quad m_{z'}(\perp, \iota\, \circledast) \qquad \Box$$

Functor $K(Y, X) \stackrel{\mathsf{def}}{=} Y \Rightarrow \mathsf{bool}$

corresponding to one method equal : **Self** × **Self** ⟶ bool

***K*–bisimulation** $R \subseteq X \times X \quad c : X \longrightarrow X \Rightarrow \mathsf{bool}$

$$\forall x, x', y, y' : X \,.\, x\,R\,x' \;\wedge\; y\,R\,y' \implies c\,x\,y \;=\; c\,x'\,y'$$

Defining a *K*–coalgebra $X \stackrel{\mathsf{def}}{=} \{x_1, x_2\}$

$c \;:\; X \longrightarrow X \Rightarrow \mathsf{bool} \qquad c\,x\,y \stackrel{\mathsf{def}}{=} \text{if } x = y \text{ then } \top \text{ else } \bot \text{ endif}$

- $\mathrm{Eq}(X) = \{(x_1, x_1),\ (x_2, x_2)\}$ is a bisimulations for c
- $R \subseteq X \times X \stackrel{\mathsf{def}}{=} \{(x_1, x_2), (x_2, x_1)\}$ is a bisimulation for c
- However, $\mathrm{Eq}(X) \cup R = X \times X$ is not a bisimulation

$$\top \quad = \quad c\,x_1\,x_1 \quad \neq \quad c\,x_1\,x_2 \quad = \quad \bot$$

Functor $G(Y, X) \stackrel{\mathsf{def}}{=} (\mathbb{N} \Rightarrow Y) \Rightarrow \mathsf{bool}$

Defining a G–coalgebra on $\mathbb{N}$

$$c \;:\; \mathbb{N} \longrightarrow (\mathbb{N} \Rightarrow \mathbb{N}) \Rightarrow \mathsf{bool} \qquad\qquad c\,x\,f \;=\; \begin{cases} \top & \exists n\,.\,\forall i\,.\,f\,i < n \\ \bot & \text{otherwise} \end{cases}$$

G–bisimulation for c $\quad R \subseteq \mathbb{N} \times \mathbb{N}$

$$\forall x, y : \mathbb{N},\ f, g : \mathbb{N} \longrightarrow \mathbb{N}\,.\, x\,R\,y \;\wedge\; \big(\forall n\,.\,(f\,n)\,R\,(g\,n)\big) \implies c\,x\,f \;=\; c\,y\,g$$

Family $(S_n)_{n \in \mathbb{N}}$ $\qquad i\,S_n\,j \iff i = j \;\vee\; (i < n \wedge j < n)$

- all S_n are bisimulations
- every *finite* subset of S_n's has an upper bound (which is a bisimulation)
- $\bigcup_n S_n \;=\; \mathbb{N} \times \mathbb{N}$ is *not* a bisimulation, take $f_0 : \mathbb{N} \longrightarrow \mathbb{N}$ with $f_0\,n = 0$ and $\mathsf{id}_{\mathbb{N}}$: $\forall n\,.\,(n, 0) \in \bigcup_n S_n$ but

$$\top \;=\; c\,0\,\mathsf{id}_{\mathbb{N}} \;\neq\; c\,0\,f_0 \;=\; \bot$$

[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.

Applications

- MSMIE (multiprocessor shared-memory information exchange) protocol
- memory management in the FIASCO micro kernel operating system

Future work

- CCSL is not actively developed currently
- Application in the VFiasco project — Verification of Fiasco

[1] H. Tews, H. Härtig, and M. Hohmuth. VFiasco — towards a provably correct μ–kernel. Technical Report TUD-FI01-1 – January 2001, TU Dresden, 2001.

[2] M. Hohmuth, H. Tews, and S. G. Stephens. Applying source-code verification to a microkernel — the VFiasco project (extended abstract). In *Proceedings of the Tenth ACM SIGOPS European Workshop*, September 2002.

Visit `http://www.vfiasco.org`

- CCSL Version 3 (brainstorming)
  - general specifications with loose/initial/final semantics; multiple sorts
  - partial operations
  - generic specifications, signature morphisms

[AM89] P. Aczel and P. F. Mendler. A final coalgebra theorem. In D. H. Pitt, D. E. Rydeheard, P. Dybjer, A. M. Pitts, and A. Poigné, editors, *Proceedings of the Conference on Category Theory and Computer Science*, volume 389 of *Lecture Notes in Computer Science*, pages 357–365, Berlin, September 1989. Springer.

[Hen99] U. Hensel. *Definition and Proof Principles for Data and Processes*. PhD thesis, Univ. of Dresden, Germany, 1999.

[HJ98] C. Hermida and B. Jacobs. Structural induction and coinduction in a fibrational setting. *Information and Computation*, 145(2):107–152, 1998.

[JT01] B. Jacobs and H. Tews. Assertional and behavioural refinement in coalgebraic specification. Submitted, 2001.

[Tew01] H. Tews. Coalgebras for binary methods: Properties of bisimulations and invariants. *Theoretical informatics and applications*, 35(1):83–111, February 2001.

[Tew02a] H. Tews. *Coalgebraic Methods for Object-Oriented Specification*. PhD thesis, University of Dresden, 2002. available via wwwtcs.inf.tu–dresden/∼tews/PhD.

[Tew02b] H. Tews. Greatest bisimulations for binary methods. In L. Moss, editor, *Coalgebraic Methods in Computer Science '02*, volume 65.1 of *ENTCS*. Elsevier, Amsterdam, 2002.

Consider a higher-order polynomial functor H and a coalgebras $c : X \longrightarrow H(X, X)$.

Invariants according to Hermida & Jacobs

- P is an invariant for c, if

$$\forall x \in X \,.\, P(x) \quad \text{implies} \quad \mathrm{Pred}(H)(P, P)(c(x))$$

Invariants according to Aczel & Mendler

$$\begin{tikzcd}
P \subseteq X \arrow[rr, "\iota"] \arrow[d] & & X \arrow[d, "c"] \\
H(P, P) \arrow[dr, "H(P,\iota)"'] & & H(X, X) \arrow[dl, "H(\iota, X)"] \\
& H(P, X) &
\end{tikzcd}$$

$$\boxed{\begin{array}{c}\text{Both notions coincide for extended polynomial functors}\\ \text{(in general they are different)}\end{array}}$$

Predicate lifting is an operation on predicates

$$\big(Q \subseteq Y,\ \ P \subseteq X\big) \quad \longmapsto \quad \mathsf{Pred}(H)(Q,P) \quad \subseteq \quad H(Y,X)$$

Defined by induction on the structure of H:

H	$\mathsf{Pred}(H)(Q,P)(a)$ for $a \in H(Y,X)$
X	$P(a)$
A	$\top$
$H_1 \times H_2$	$\mathsf{Pred}(H_1)(Q,P)(\pi_1\, a) \ \wedge\ \mathsf{Pred}(H_2)(Q,P)(\pi_2\, a)$
$H_1 + H_2$	$\begin{cases} \text{if } a = \kappa_1\, a_1 & \mathsf{Pred}(H_1)(Q,P)(a_1) \\ \text{if } a = \kappa_2\, a_2 & \mathsf{Pred}(H_2)(Q,P)(a_2) \end{cases}$
$H_1 \Rightarrow H_2$	$\forall u \in \mathsf{Pred}(H_1)(P,Q)\ .\ \mathsf{Pred}(H_2)(Q,P)(a(u))$