CCSL: Coalgebras meet Algebras meet Higher-Order Logic

Hendrik Tews

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

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

$=$ CCSL

Basics

Special Features

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

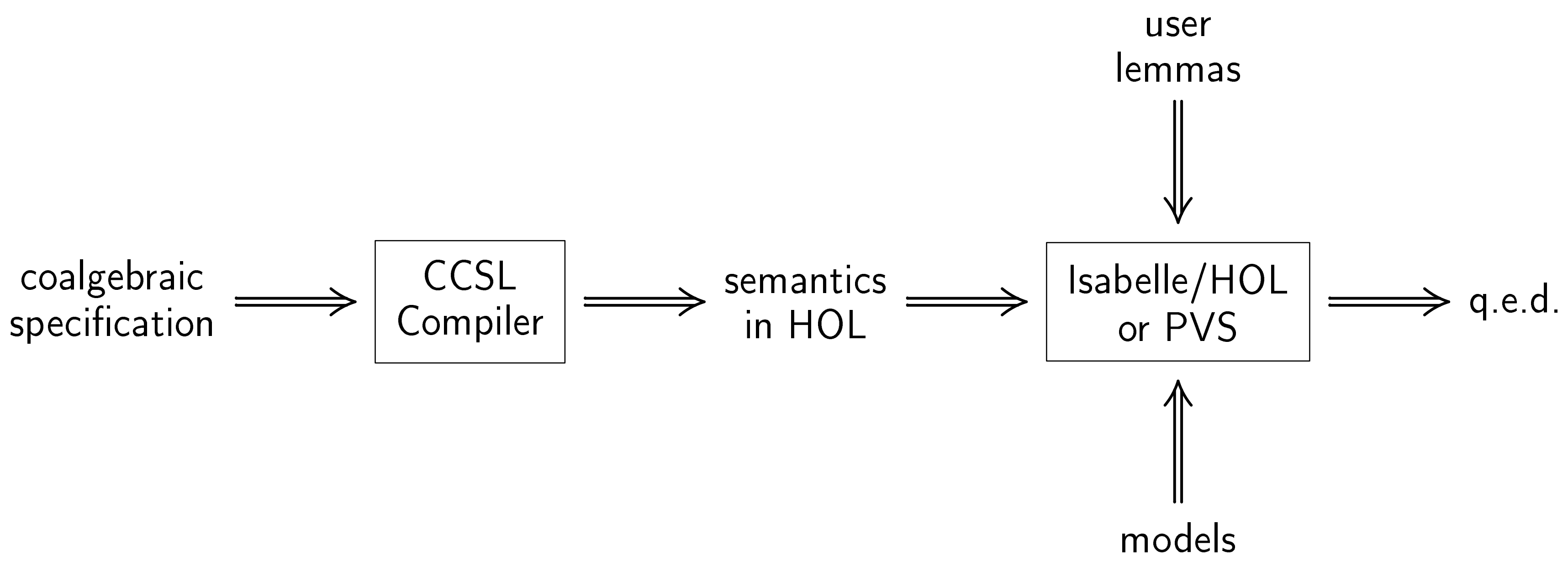
$$T(X) = \mathbb{R} \times \mathbb{R} \times (\mathbb{R} \times \mathbb{R} \Rightarrow X)$$

---

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
```

```
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
- Higher-order iterated polynomial functors (including finite/non-finite powerset)
- ADT's and Classes can be nested $\Longrightarrow$ nested induction and coinduction
- higher-order logic
- support for PVS and Isabelle/HOL
- coalgebraic refinement [JT01]
- be permissive: allow the user to shoot himself
- Version 2.2.7 available at http://wwwtcs.inf.tu-dresden.de/~tews/ccsl

[1] 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.

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

[3] H. Tews. The coalgebraic class specification language CCSL: Syntax and semantics. Technical Report TUD–FI02–08–August 2002, Dresden University of Technology, 2002. Available from `wwwtcs.inf.tu-dresden.de/∼tews/science.html`.

[4] H. Tews. The coalgebraic class specification language CCSL: Tutorial. Draft version available from `wwwtcs.inf.tu-dresden.de/∼tews/ccsl`, 2004.

Finitely branching trees of finite depth

Begin FinTreeFin[A : **Type**] : **Adt**
  Constructor
    next : list[[A, **Carrier**]] –> **Carrier**
End FinTreeFin

- standard semantics via initial algebras
- adt's are translated into adt's of the target theorem prover
- compiler generates accessors, recognizers,
  predicate and relation lifting, the map combinator, and reduce

[1] S. Owre and N. Shankar. Abstract datatypes in PVS. Technical Report SRI-CSL-93-9R, Computer Science Laboratory, SRI International, Menlo Park, CA, December 1993. Extensively revised June 1997; Also available as NASA Contractor Report CR-97-206264.

[2] S. Berghofer and M. Wenzel. Inductive datatypes in HOL — lessons learned in formal-logic engineering. In Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin, and L. Théry, editors, *Theorem Proving in Higher Order Logics*, volume 1690 of *Lecture Notes in Computer Science*, pages 19–36. Springer, September 1999.

- single sorted coalgebraic specifications (**Self** stands for the distinguished sort)
- Signatures are modelled by functors
- CCSL builts on a enriched version of $\lambda\!\rightarrow$
- specifications can have type parameters
- specifications and type declarations introduce new type constructors
- type constructors are collected in a ground signature

Types

type	concreate syntax
Self	**Self**
$\sigma \times \tau$, $\sigma + \tau$, $\sigma \Rightarrow \tau$	[A,B], Coproduct[A,B], [A –> B]
$\mathsf{C}[\cdots]$	C[. . .]
$\nu X\,.\,\tau$	Adt specification
$\mu X\,.\,\tau$	Class specification

- semantics of types builts on variance checking:
  for each type variable determine if it occurs positively or negatively
- different external and internal semantics of types
- Externally: a type τ with n type parameters is translated literally into a theory with $n+1$ type parameters (**Self** becomes a type parameter)
- Internally τ defines a mixed-variance functor $[\![\tau]\!] : \mathbf{Set}^{2n+2} \longrightarrow \mathbf{Set}$
  (every type variable positively and negatively, plus self)
- messy details for mixed variance type constructors left out

Example

$$\begin{aligned} \tau \quad &= \quad A \Rightarrow (\textsf{\textbf{Self}} + B) \\ [\![\tau]\!] \quad &: \quad \mathbf{Set}^6 \longrightarrow \mathbf{Set} \\ [\![\tau]\!](U^-, U^+, V^-, V^+, X^-, X^+) \quad &= \quad U^- \Rightarrow (X^+ \uplus V^+) \end{aligned}$$

- consider a method declaration $m : \mathbf{Self} \times \sigma \longrightarrow \tau$ with n type parameters
- fix an interpretation $U_1, \ldots, U_n$ for the type parameters
- set $H(Y, X) = [\![\sigma \Rightarrow \tau]\!](U_1, U_1, \ldots, U_n, U_n, Y, X)$
- H–coalgebras are used as semantics of m

Coalgebras for Mixed-variance Functors

- Coalgebras are morphisms $X \xrightarrow{c} H(X, X)$
- Morphism of coalgebras:

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

- see [Tew01, Tew02b, Tew02a]
- If **Self** occurs only positively in $\sigma \Rightarrow \tau$ then everything collapses to normal

Consider a specification $\mathcal{S}$ with

method declarations	$m_i : \textsf{\textbf{Self}} \times \rho_i \longrightarrow \tau_i$	yield $\tau_{\mathcal{S}} = \cdots \times (\rho_i \Rightarrow \tau_i) \times \cdots$
constructor declarations	$c_j : \sigma_j \longrightarrow \textsf{\textbf{Self}}$	yield $\sigma_{\mathcal{S}} = \cdots + \sigma_j + \cdots$
assertions	$x_k : \textsf{\textbf{Self}} \vdash \varphi_k$	yield $\mathcal{S}_M = \bigwedge \varphi_k$
creation conditions	$\emptyset \vdash \psi_l$	yield $\mathcal{S}_C = \bigwedge \psi_l$

Model of signature: A set X with two functions

$$[\![\sigma_{\mathcal{S}}]\!] \xrightarrow{\quad a \quad} X \xrightarrow{\quad c \quad} [\![\tau_{\mathcal{S}}]\!](X, X)$$

interpreting all operations from the specification.

Model: $\langle X, c, a \rangle$ is a model of $\mathcal{S}$, if

$$\forall x \in X \,.\, [\![\mathcal{S}_M]\!]_c(s) \qquad \text{and} \qquad a \in [\![\mathcal{S}_C]\!]_c$$

Parameters: lead to indexing: $\langle X, c, a \rangle_{U_1 \ldots, U_n}$

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

Invariants according to Hermida & Jacobs [HJ98]

- $P \subseteq X$ is an invariant for c, if $\forall x \in X \,.\, P(x)$ implies $\mathrm{Pred}(H)(P, P)(c(x))$
- P is a strong invariant for c, if $\forall x \in X \,.\, P(x)$ implies $\mathrm{Pred}(H)(\top_X, P)(c(x))$

Invariants according to Aczel & Mendler [AM89]

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

Hermida/Jacobs (simple) invariants and Aczel/Mendler invariants coincide for extended polynomial functors (in general they are different)

Predicate lifting is an operation on predicates

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

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

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

Extending polynomial functors

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

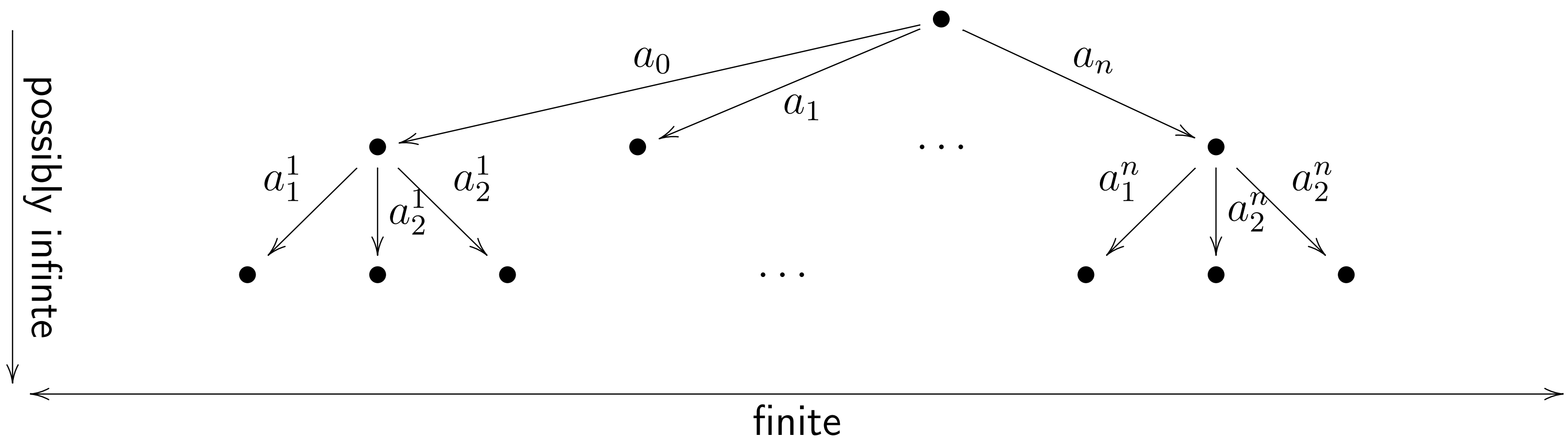
Extension

polynomial	• structural properties • bisimulations/invariants form a *complete* lattice • greatest bisimulations/invariants • final models
extended cartesian	• structural properties • bisimulations form an *incomplete* lattice • bisimulation equivalences form a *complete* lattice • greatest bisimulations equivalences for all coalgebras
extended polynomial	• structural properties • bisimulations form an *incomplete* lattice • greatest bisimulations equivalences for finitely based coalgebras
higher-order	• strong invariants form a complete lattice • otherwise only trivial properties

- Results/Examples have been checked/developed with PVS.
  see `http://wwwtcs.inf.tu-dresden.de/~tews/binary/`

```
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 $\mathsf{next}(\bot) = \mathsf{null}$

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)

[1] U. Hensel and B. Jacobs. Proof principles for datatypes with iterated recursion. In E. Moggi and G. Rosolini, editors, *Category Theory and Computer Science*, volume 1290 of *Lecture Notes in Computer Science*, pages 220–241. Springer, Berlin, 1997.

Invariant is a predicate closed under the coalgebra

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

Strong invariants form a complete lattice

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

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

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)

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

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

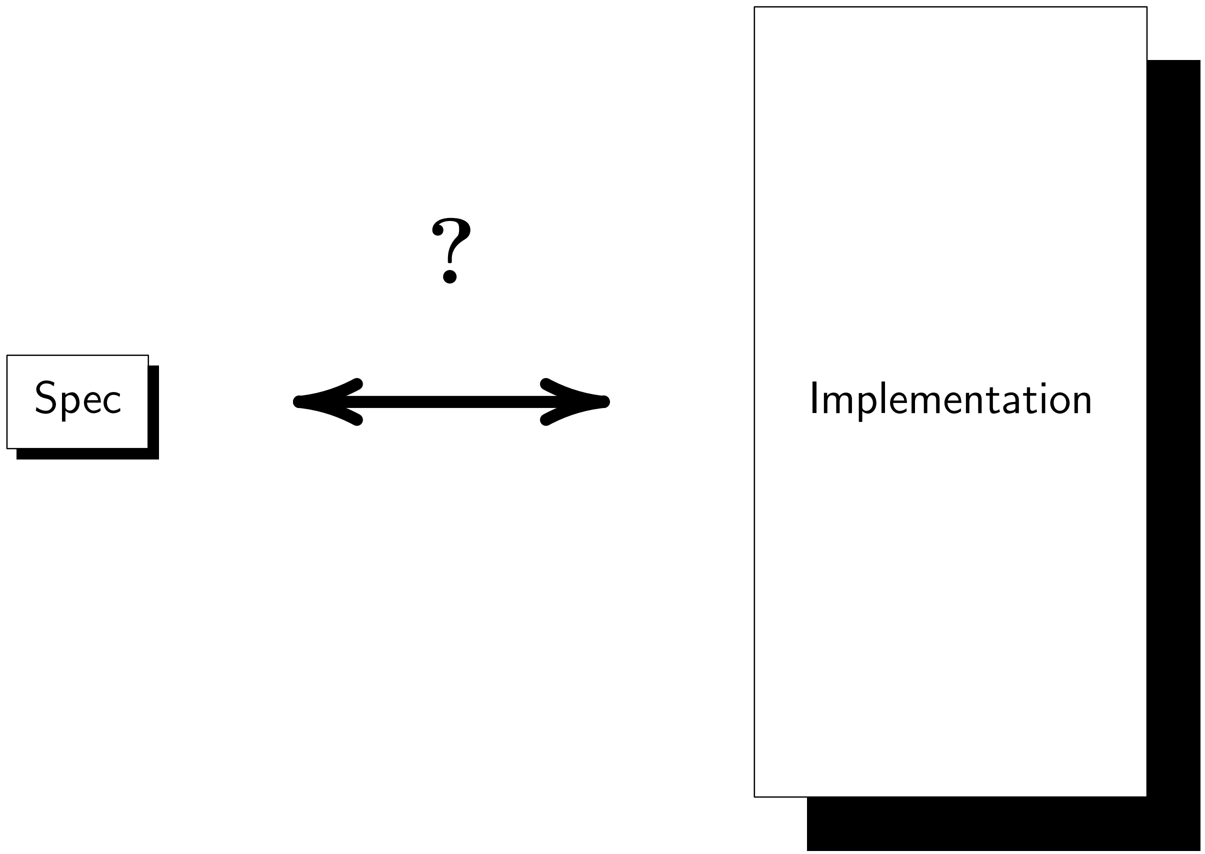
Semantics

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

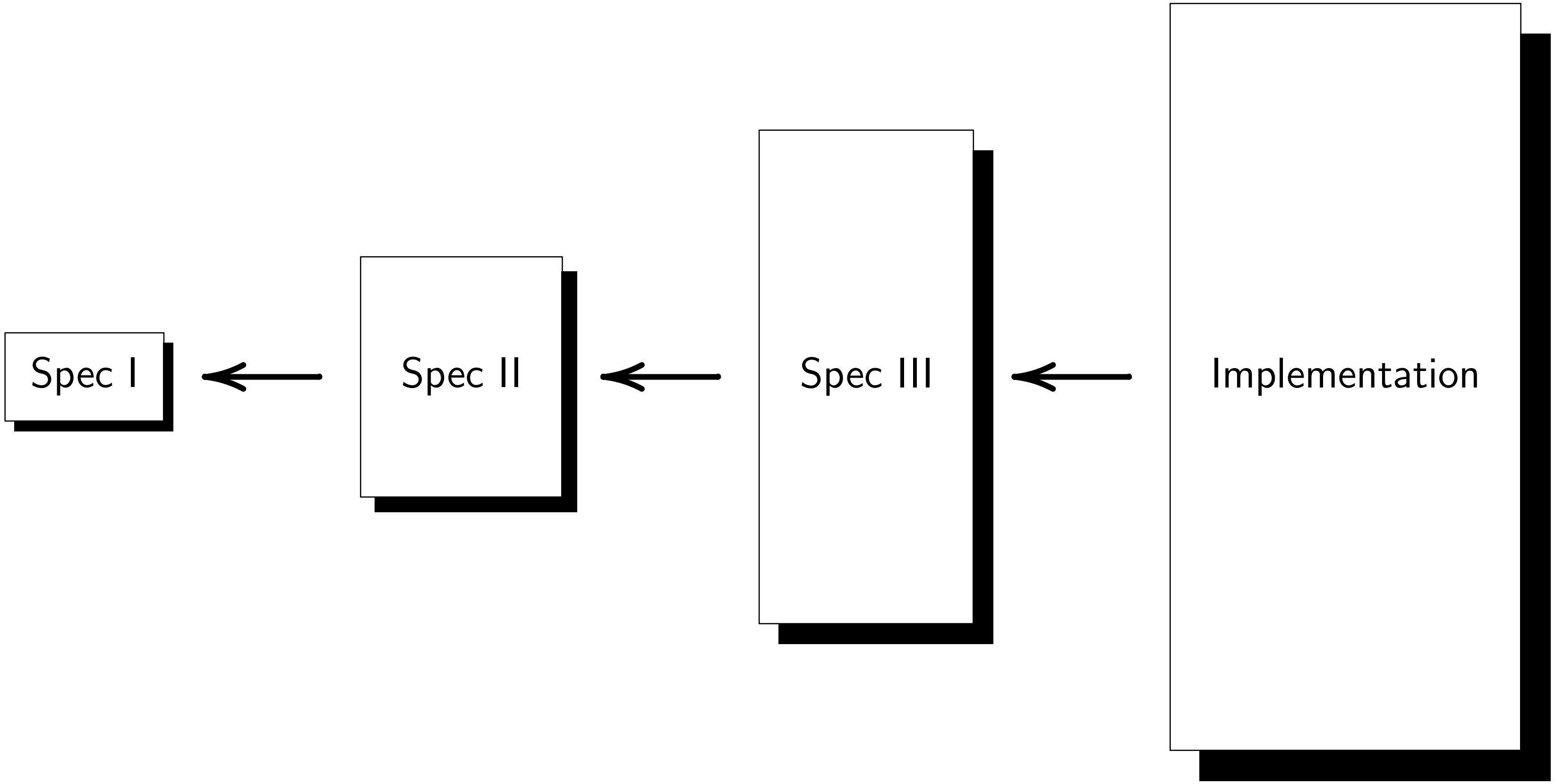
Eventually $\Diamond\varphi \stackrel{\mathsf{def}}{=} \neg\Box\neg\varphi \qquad \Big(\stackrel{\mathsf{def}}{=} \lambda x : \mathbf{Self}\,.\,\neg(\Box\lambda y : \mathbf{Self}\,.\,\neg\varphi(y))\,(x)\Big)$

Method-wise modal operators arise from subcoalgebras

Problem



Using refinement



Formalization “Spec II” *refines* “Spec I” if

M is a model of “Spec II” $\Longrightarrow$ M is also a model of “Spec I”

Consider an "abstract" specification $\mathcal{A}$ and a "concrete" specification $\mathcal{B}$ with:

- signatures $\Sigma_{\mathcal{A}} = \langle \tau_{\mathcal{A}}, \sigma_{\mathcal{A}} \rangle$ and $\Sigma_{\mathcal{B}} = \langle \tau_{\mathcal{B}}, \sigma_{\mathcal{B}} \rangle$
- assertions $\mathcal{A}_M$, $\mathcal{B}_M$ and
- creation conditions $\mathcal{A}_C$, $\mathcal{B}_C$

The specification $\mathcal{B}$ is an *assertional refinement* of $\mathcal{A}$, just in case

for every model $[\![\sigma_{\mathcal{B}}]\!](S) \xrightarrow{a} S \xrightarrow{c} [\![\tau_{\mathcal{B}}]\!](S)$ of $\mathcal{B}$

there exists $[\![\sigma_{\mathcal{A}}]\!](S) \xrightarrow{a'} S \xrightarrow{c'} [\![\tau_{\mathcal{A}}]\!](S)$

such that $\langle \overline{a'}, c', a' \rangle$ is a model of $\mathcal{A}$.

$\left(\begin{array}{l} \overline{a'} \text{ denotes the least invariant, that contains } a' \\ \text{Corresponds to the set of reachable states.} \end{array} \right)$

Assertional Refinement

- a *parameter translation* deals with different parameters of the abstract and the concrete specification (in most interesting cases it is the identity)
- requires to prove the assertions of the abstract specification
- thus requires refining both public *and privat* methods

Behavioural Refinement

- requires behavioural equivalence between the (translated) concrete model and some abstract model
- permits Refinement with respect to a subsignature of $\Sigma_{\mathcal{A}}$
- thus private methods can be omitted
- more general than assertional refinement
- coincides under reasonable assertional in the common case

[1] B. Jacobs and H. Tews. Assertional and behavioural refinement in coalgebraic specification. Available from `wwwtcs.inf.tu-dresden.de/~tews/science.html`, 2001.

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

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
```

- implemented in Ocaml
- 60,000 lines of code, 150 files
- 13 passes:
    1. lexing
    2. parsing (built symboltable, resolve type identifiers)
    3. create update methods
    4. resolve inheritance (and renaming)
    5. generate update assertions
    6. collect component and features
    7. generate coreduce & friends
    8. identifier resolution
    9. typechecking
    10. pedantic checks
    11. generate output in internal enriched HOL
    12. convert some CCSL specifics down to plain HOL
    13. pretty print

[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.

From the transaction case study [JT01]

```
Begin TransactionCommitAbstract : ClassSpec
  Method
    val    : Self -> int;                              % current value
    add    : [Self, int] -> InterruptOrSucceed[Self];  % the action
    start  : Self -> Self;                             % start of transaction
    commit : Self -> InterruptOrSucceed[Self];         % end of transaction
  ....
Begin TransactionState : ClassSpec
  Private Method
      val        : Self -> int;                        % value field
      set_val    : [Self, int] -> Self;                % set value field
      args       : Self -> list[int];                  % intended adds
      add_arg    : [Self,int] -> [OkOrFail,Self];      % cons to args field
      reset_args : Self -> Self;                       % set args to nil
```

describes coalgebras for the functors

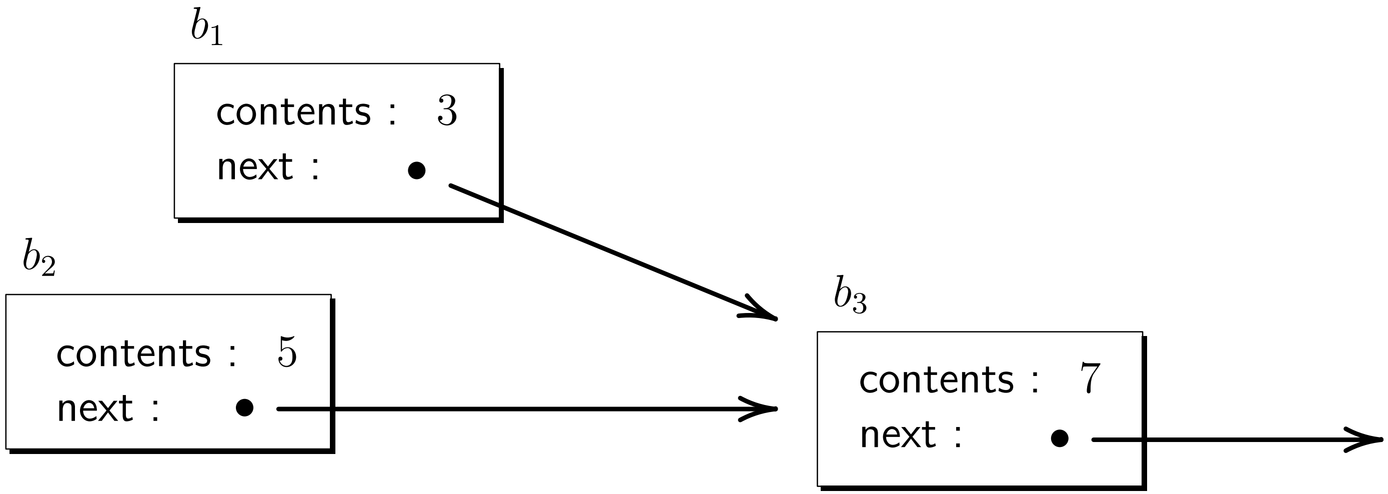
$$\begin{aligned} T_A(X) &= \mathsf{int} \times (\mathsf{int} \Rightarrow X + X) \times X \times (X + X) \\ T_S(X) &= \mathsf{int} \times (\mathsf{int} \Rightarrow X) \times \mathsf{list}[\mathsf{int}] \times (\mathsf{bool} \times X) \times X \end{aligned}$$

in Coq (or PVS CoDatatype):

```
CoInductive Set TA := c : int => (int => TA + TA) => TA => (TA + TA) => TA.
CoInductive Set TS := c : int => (int => TS) =>
                          list int => (bool × TS) => TS => TS.
```

Comparison with Hidden algebra

- bisimulations / invariants automagically (all operations are congruent)
- bisimulations relate *different models* with *different state space*
- structured types



Begin ChainedBuf[I : **Type**] : **ClassSpec**
  Method
    store : [**Self**, I] –> **Self**;
    get : **Self** –> lift[I];

    push : [**Self**, [nat –> lift[**Self**]]] –> [**Self**, [nat –> lift[**Self**]]];
    . . .

Applications

- MSMIE (multiprocessor shared-memory information exchange) protocol
- memory management in the FIASCO micro kernel operating system
- The Coinductive Approach to Verifying Cryptographic Protocols [HW03]

Future work

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

  [1] 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 ideas: partial operations, full algebraic specifications, signature morphisms

Conclusion

- algebras and coalgebras combined with HOL yield a powerful specification language
- "traditional coalgebra" is not sufficient – need mixed variance functors
- traditional bisimulations/invariants are not practical, sometimes even wrong

[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.

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

[HW03] J. Hughes and M. Warnier. The Coinductive Approach to Verifying Cryptographic Protocols. In M. Wirsing, D. Pattinson, and R. Hennicker, editors, *Recent Trends in Algebraic Development Techniques*, volume 2755 of *Lecture Notes in Computer Science*, pages 268–283. Springer, Berlin, 2003.

[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.