Coalgebraic Perspectives on Abstract State Machines

Daniel Schreckling Eric Rothstein

IT-Security Group
University of Passau
Passau, Germany

BIOMICS Summer Workshop
June 2014

Coalgebraic and Monadic Perspectives on Abstract State Machines

Daniel Schreckling Eric Rothstein

IT-Security Group
University of Passau
Passau, Germany

BIOMICS Summer Workshop
June 2014

Research Agenda in BIOMICS

(Partial) Goal

- Definition and implementation of an interaction machine
- Specification of computation through *behaviour*
- Machine and specifications reflect construction principles known from bio-chemical systems

Research Agenda in BIOMICS

(Partial) Goal

- Definition and implementation of an interaction machine
- Specification of computation through *behaviour*
- Machine and specifications reflect construction principles known from bio-chemical systems

Research Steps (Backup plan)

Research Agenda in BIOMICS

(Partial) Goal

- Definition and implementation of an interaction machine
- Specification of computation through *behaviour*
- Machine and specifications reflect construction principles known from bio-chemical systems

Research Steps (Backup plan)

- Describe structures and dynamics of systems using (co)algebra
- Find a specification language which can be modelled using (co)algebra
- Use category theory to map (co)algebras of system to (co)algebraic properties of the language

ASM: Candidate for the specification language in BIOMICS

ASM: Candidate for the specification language in BIOMICS

Requirements

- Understanding of the modelling methodology and formalities
- **Mapping structural and dynamical ASM properties to (co)algebras**
- Defining Links between other system (from biology) and ASMs
- Refine expressive power of ASMs to a language we need

Contents

Motivation

- Main Idea of Abstract State Machines
  - Combines developments of formal logic in decades
  - Tarksi: structures, including functions and predicates over real world items (most general mathematical framework)
  - First order logic: developed to define and analyze structures
  - ASMs are direct consequence: introduce algorithms on real world objects

Motivation

- Main Idea of Abstract State Machines
  - Combines developments of formal logic in decades
  - Tarksi: structures, including functions and predicates over real world items (most general mathematical framework)
  - First order logic: developed to define and analyze structures
  - ASMs are direct consequence: introduce algorithms on real world objects
- What to use it for?
  - Description of procedures involving real world items
  - Specifying steps of dynamic, discrete systems
  - Verification of software models
  - ...

Motivation

- Main Idea of Abstract State Machines
  - Combines developments of formal logic in decades
  - Tarksi: structures, including functions and predicates over real world items (most general mathematical framework)
  - First order logic: developed to define and analyze structures
  - ASMs are direct consequence: introduce algorithms on real world objects
- What to use it for?
  - Description of procedures involving real world items
  - Specifying steps of dynamic, discrete systems
  - Verification of software models
  - ...
- Why Abstract States?
  - Systems are described using different levels of abstraction
  - Details and semantics of states determine abstraction

Abstract States — A Definition

- Let Υ denote a finite **Vocabulary** of tuples (f, k)
  - f are names (character sequences) of functions and relations
  - k specifies their arity f
  - $(true, 0)$, $(false, 0)$, $(\perp, 0)$, $(Boole, 0)$ are in every Υ
  - Boolean operators and equality sign are in Υ

Abstract States — A Definition

- Let Υ denote a finite **Vocabulary** of tuples (f, k)
  - f are names (character sequences) of functions and relations
  - k specifies their arity f
  - $(true, 0)$, $(false, 0)$, $(\perp, 0)$, $(Boole, 0)$ are in every Υ
  - Boolean operators and equality sign are in Υ
- Let $\mathcal{T}(\Upsilon)$ denote a set of **Terms**
  - if $(f, 0) \in \Upsilon$, then $f \in T(\Upsilon)$
  - if $(f, k) \in \Upsilon$, $t_1, \ldots, t_k \in T(\Upsilon)$, then $f(t_1, \ldots, t_k) \in \mathcal{T}(\Upsilon)$.

Abstract States — A Definition

- Let Υ denote a finite **Vocabulary** of tuples (f, k)
  - f are names (character sequences) of functions and relations
  - k specifies their arity f
  - $(true, 0)$, $(false, 0)$, $(\bot, 0)$, $(Boole, 0)$ are in every Υ
  - Boolean operators and equality sign are in Υ
- Let $\mathcal{T}(\Upsilon)$ denote a set of **Terms**
  - if $(f, 0) \in \Upsilon$, then $f \in T(\Upsilon)$
  - if $(f, k) \in \Upsilon$, $t_1, \ldots, t_k \in T(\Upsilon)$, then $f(t_1, \ldots, t_k) \in \mathcal{T}(\Upsilon)$.
- Let $\mathcal{S}(\Upsilon)$ denote the set of all **Structures** S where S satisfies
  - it has a fixed *base set* X
  - it has an interpretation $\mathcal{I}_S$ for functions/terms in Υ

Abstract States — A Definition

- Let Υ denote a finite **Vocabulary** of tuples (f, k)
  - f are names (character sequences) of functions and relations
  - k specifies their arity f
  - $(\mathit{true}, 0)$, $(\mathit{false}, 0)$, $(\perp, 0)$, $(\mathit{Boole}, 0)$ are in every Υ
  - Boolean operators and equality sign are in Υ
- Let $\mathcal{T}(\Upsilon)$ denote a set of **Terms**
  - if $(f, 0) \in \Upsilon$, then $f \in T(\Upsilon)$
  - if $(f, k) \in \Upsilon$, $t_1, \ldots, t_k \in T(\Upsilon)$, then $f(t_1, \ldots, t_k) \in \mathcal{T}(\Upsilon)$.
- Let $\mathcal{S}(\Upsilon)$ denote the set of all **Structures** S where S satisfies
  - it has a fixed *base set* X
  - it has an interpretation $\mathcal{I}_S$ for functions/terms in Υ
- **States** are described by (first-order) structures

Updates Transit Between Abstract States

- Primitive updates
  - consider states S as memory space
  - if $(f, k) \in \Upsilon$ and the k-tuple $\overline{a}$ has elements from base set X, $(f, \overline{a})$ is a **location**
  - let $(f, \overline{a}, b)$ denote an **update** of location $\overline{a}$ with b
  - change in memory Δ by update u is denoted by $\Delta(u, S)$

Updates Transit Between Abstract States

- Primitive updates
  - consider states S as memory space
  - if $(f, k) \in \Upsilon$ and the k-tuple $\overline{a}$ has elements from base set X, $(f, \overline{a})$ is a **location**
  - let $(f, \overline{a}, b)$ denote an **update** of location $\overline{a}$ with b
  - change in memory Δ by update u is denoted by $\Delta(u, S)$
- Update rules
  - terms as arguments allow for programming with updates
  - update rules R of vocabulary Υ have the form $f(t_1, ..., t_k) := t_0$ with k-ary function f and terms $t_0, ..., t_k$
  - firing updates now requires evaluating the terms in a state S

Updates Transit Between Abstract States

- Primitive updates
  - consider states S as memory space
  - if $(f, k) \in \Upsilon$ and the k-tuple $\overline{a}$ has elements from base set X, $(f, \overline{a})$ is a **location**
  - let $(f, \overline{a}, b)$ denote an **update** of location $\overline{a}$ with b
  - change in memory Δ by update u is denoted by $\Delta(u, S)$
- Update rules
  - terms as arguments allow for programming with updates
  - update rules R of vocabulary Υ have the form $f(t_1, \dots, t_k) := t_0$ with k-ary function f and terms $t_0, \dots, t_k$
  - firing updates now requires evaluating the terms in a state S
- Parallel updates are grouped in a `par` rule R
  - allows to group multiple update rules $R_1, \dots, R_j$
  - change of memory in S: $\Delta(R, S) = \Delta(R_1, S) \cup \dots \cup \Delta(R_j, S)$

Conditional Rules and Programs for ASMs

- Conditional rule R
  - Boolean term ϕ over Υ
  - Rules R_1, R_2 in Υ

    $$\texttt{if}\ \phi\ \texttt{then}\ R_1\ \texttt{else}\ R_2\ \texttt{endif}$$

  - Common semantics, i.e. $\Delta(R, S) = \Delta(R_1, S)$ if ϕ valuates to `true` in S, $\Delta(R_2, S)$ otherwise

Conditional Rules and Programs for ASMs

- Conditional rule R
  - Boolean term ϕ over Υ
  - Rules R_1, R_2 in Υ

    `if` ϕ `then` R_1 `else` R_2 `endif`

  - Common semantics, i.e. $\Delta(R, S) = \Delta(R_1, S)$ if ϕ valuates to `true` in S, $\Delta(R_2, S)$ otherwise
- Program Π over Υ
  - Π is a rule over Υ
  - $\Delta(\Pi, S)$ is well defined for every state S defined by Υ
  - changes applied to S by Π are defined by $\tau_\Pi(S) = S + \Delta(\Pi, S)$

A Small Example for an ASM Rule/Program

- Input: $a, b \in \mathbb{N}$
- Otutput: $d = gcd(a, b)$
- A single step in the Euclidean Algorithm can be described by

```
if      b = 0 then d := a
else if b = 1 then d := 1
else
  par
    a := b
    b := a mod b
  endpar
endif
```

$(a = 12, b = 6, d = \perp)_{S_1}$

BIOMICS

A Small Example for an ASM Rule/Program

- Input: $a, b \in \mathbb{N}$
- Otutput: $d = gcd(a, b)$
- A single step in the Euclidean Algorithm can be described by

```
if      b = 0 then d := a
else if b = 1 then d := 1
else
  par
    a := b
    b := a mod b
  endpar
endif
```

$(a = 12, b = 6, d = \perp)_{S_1} \rightarrow (6, 0, \perp)_{S_2}$

A Small Example for an ASM Rule/Program

- Input: $a, b \in \mathbb{N}$
- Otutput: $d = gcd(a, b)$
- A single step in the Euclidean Algorithm can be described by

```
if      b = 0 then d := a
else if b = 1 then d := 1
else
  par
    a := b
    b := a mod b
  endpar
endif
```

$$(a = 12, b = 6, d = \perp)_{S_1} \rightarrow (6, 0, \perp)_{S_2} \rightarrow (6, 0, 6)_{S_3}$$

Abstract State Machine (ASM)

Definition

An (sequential) ASM M is denoted by $M = (\Upsilon, \Pi, \mathcal{B}, \mathcal{S}, \mathcal{I}, \tau)$

- Vocabulary Υ
- Program Π over Υ
- Base set $\mathcal{B}$
- Set of abtract states $\mathcal{S}(\Pi)$
- Set of initial states $\mathcal{I}(\Pi) \subset \mathcal{S}(\Pi)$
- transition function τ_Π

Coalgebras — A Definition

Definition (F-Coalgebras)

Given are a catgeory $\mathcal{C} = (O, M, s, t, c)$ and functor $F\colon \mathcal{C} \to \mathcal{C}$. $\mathbb{C} = \langle C, \tau \rangle_F$ is called F-coalgebra, if $C \in O$, and $\tau \in M$ with $\tau\colon C \to F(C)$. C is called *carrier 'set'* (or state space) and τ is called *structure* (or transition) of coalgebra $\mathbb{C}$.

Coalgebras Structure Codomains

- Imperative programs without side-effects, exceptions, non-termination, etc.

`i:=5` $\qquad\qquad S \xrightarrow{\texttt{i:=5}} S$

Coalgebras Structure Codomains

- Imperative programs without side-effects, exceptions, non-termination, etc.

`i:=5` $\qquad$ $S \xrightarrow{\texttt{i:=5}} S$

- Extend state set to account for non-termination by introducing new symbol $\perp$ with $S_\perp = S \cup \{\perp\}$

`statement` $\qquad$ $S_\perp \xrightarrow{\texttt{statement}} S_\perp$

Coalgebras Structure Codomains

- Imperative programs without side-effects, exceptions, non-termination, etc.

`i:=5` $\qquad S \xrightarrow{\texttt{i:=5}} S$

- Extend state set to account for non-termination by introducing new symbol $\perp$ with $S_\perp = S \cup \{\perp\}$

`statement` $\qquad S_\perp \xrightarrow{\texttt{statement}} S_\perp$

- Account for exceptions E
$S \cup \{\perp\} \cup (S \times E) \xrightarrow{\texttt{statement}} S \cup \{\perp\} \cup (S \times E)$

Coalgebras Structure Codomains

- Imperative programs without side-effects, exceptions, non-termination, etc.

`i:=5` $\qquad S \xrightarrow{\texttt{i:=5}} S$

- Extend state set to account for non-termination by introducing new symbol $\perp$ with $S_\perp = S \cup \{\perp\}$

`statement` $\qquad S_\perp \xrightarrow{\texttt{statement}} S_\perp$

- Account for exceptions E
$S \cup \{\perp\} \cup (S \times E) \xrightarrow{\texttt{statement}} S \cup \{\perp\} \cup (S \times E)$

Question

What if we kept a constant state set and structure the codomain?

Coalgebras are about Observations

- Consider sequences A^∞ over a set A
  - finite sequences $A^* = \langle a_1, a_2, \ldots, a_n \rangle$
  - infinite sequences $A^{\mathbb{N}} = \langle a_1, a_2, \ldots \rangle$
  - all sequences $A^\infty = A^* \cup A^{\mathbb{N}}$
- Consider function *next*

$$
next : A^\infty \longrightarrow \{\perp\} \cup (A \times A^\infty) \tag{1}
$$

defined by the total function σ with symbol $\perp$

$$
\sigma \longmapsto \begin{cases} \perp & \textit{if } \sigma \text{ is the empty sequence} \\ (a, \sigma') & \textit{if } \sigma = a \cdot \sigma' \text{ with } \textit{head } a \text{ and } \textit{tail } \sigma' \end{cases} \tag{2}
$$

- Applying *next* generates all *observable* elements of A^∞

Coalgebras and Behaviours

- Properties of the function *next*
  - coalgebra of type $\{\perp\} \cup (A \times (-))$
  - terminal (final) coalgebra among all F-coalgebras with $F(S) = \{\perp\} \cup (A \times S)$

Coalgebras and Behaviours

- Properties of the function *next*
  - coalgebra of type $\{\perp\} \cup (A \times (-))$
  - terminal (final) coalgebra among all F-coalgebras with $F(S) = \{\perp\} \cup (A \times S)$
- Implications of Terminality
  - final coalgebras (objects) are unique up to isomorphism
  - there is a unique *behaviour* function $beh_c : S \to A^\infty$ for any F-coalgebra with state space S and $c : S \to \{\perp\} \cup (A \times S)$

Coalgebras and Behaviours

- Properties of the function *next*
  - coalgebra of type $\{\bot\} \cup (A \times (-))$
  - terminal (final) coalgebra among all F-coalgebras with $F(S) = \{\bot\} \cup (A \times S)$
- Implications of Terminality
  - final coalgebras (objects) are unique up to isomorphism
  - there is a unique *behaviour* function $beh_c : S \to A^\infty$ for any F-coalgebra with state space S and $c : S \to \{\bot\} \cup (A \times S)$
  - beh_c is a homomorphism of coalgebras

$$
\begin{tikzcd}
\{\bot\} \cup (A \times S) \arrow[r, dashed, "id \cup (id \times beh_c)"] & \bot \cup (A \times A^\infty) \\
S \arrow[u, "c"] \arrow[r, dashed, "beh_c"'] & A^\infty \arrow[u, "next"']
\end{tikzcd}
$$

Termial F-coalgebras and their Existence

- Terminal F-coalgebra determines behaviour for any coalgebra in category $\mathbf{CoAlg(F)}$

$$\begin{tikzcd} F(X) \arrow[r, dashed, "F(beh_c)"] & F(Z) \\ X \arrow[u, "c"] \arrow[r, dashed, "beh_c"] & Z \arrow[u, "\zeta"'] \end{tikzcd}$$

Termial F-coalgebras and their Existence

- Terminal F-coalgebra determines behaviour for any coalgebra in category **CoAlg**(**F**)

$$\begin{tikzcd} F(X) \arrow[r, dashed, "F(beh_c)"] & F(Z) \\ X \arrow[u, "c"] \arrow[r, dashed, "beh_c"] & Z \arrow[u, "\zeta"'] \end{tikzcd}$$

- Unfortunately: No guarantee that terminal F-coalgebra exists

Termial F-coalgebras and their Existence

- Terminal F-coalgebra determines behaviour for any coalgebra in category $\mathbf{CoAlg}(\mathbf{F})$

$$\begin{tikzcd} F(X) \arrow[r, dashed, "F(beh_c)"] & F(Z) \\ X \arrow[u, "c"] \arrow[r, dashed, "beh_c"] & Z \arrow[u, "\zeta"'] \end{tikzcd}$$

- Unfortunately: No guarantee that terminal F-coalgebra exists
- But: Finite Kripke polynomial functors have terminal F-coalg.
- Various non-trivial methods to find terminal coalgebras

Bisimilarity and F-coalgebras

Comparing systems using a terminal coalgebra is *simple*

Bisimilarity and F-coalgebras

Comparing systems using a terminal coalgebra is *simple*

What if the terminal coalgebra does not exist or is hard to find?

Bisimilarity and F-coalgebras

Comparing systems using a terminal coalgebra is *simple*

What if the terminal coalgebra does not exist or is hard to find?

Bisimulation is a means for comparing *behaviour*

Bisimilarity and F-coalgebras

Comparing systems using a terminal coalgebra is *simple*
What if the terminal coalgebra does not exist or is hard to find?
Bisimulation is a means for comparing *behaviour*

- **Relation lifting**
  - $F\colon \mathbf{Set} \to \mathbf{Set}$ be a polynomial functor
  - X, Y arbitrary objects in $\mathbf{Set}$
  - Relation lift sends $R \subseteq X \times Y$ to $Rel(F)(R) \subseteq F(X) \times F(Y)$

Bisimilarity and F-coalgebras

Comparing systems using a terminal coalgebra is *simple*

What if the terminal coalgebra does not exist or is hard to find?

Bisimulation is a means for comparing *behaviour*

- **Relation lifting**
  - $F\colon \mathbf{Set} \to \mathbf{Set}$ be a polynomial functor
  - X, Y arbitrary objects in **Set**
  - Relation lift sends $R \subseteq X \times Y$ to $Rel(F)(R) \subseteq F(X) \times F(Y)$
- **Bisimulation**
  - Let $c\colon X \to F(X)$ and $d\colon Y \to F(Y)$ be two coalgebras on F.
  - Relation $R \subseteq X \times Y$ describes the bisimulation for c and d which is closed under c and d:

$$\forall x \in X, y \in Y, (x, y) \in R \Rightarrow (c(x), d(y)) \in Rel(F)(R)$$

Bisimilarity and F-coalgebras

Comparing systems using a terminal coalgebra is *simple*
What if the terminal coalgebra does not exist or is hard to find?
Bisimulation is a means for comparing *behaviour*

- **Relation lifting**
  - $F\colon \mathbf{Set} \to \mathbf{Set}$ be a polynomial functor
  - X, Y arbitrary objects in **Set**
  - Relation lift sends $R \subseteq X \times Y$ to $Rel(F)(R) \subseteq F(X) \times F(Y)$
- **Bisimulation**
  - Let $c\colon X \to F(X)$ and $d\colon Y \to F(Y)$ be two coalgebras on F.
  - Relation $R \subseteq X \times Y$ describes the bisimulation for c and d which is closed under c and d:

$$\forall x \in X, y \in Y, (x, y) \in R \Rightarrow (c(x), d(y)) \in Rel(F)(R)$$

- **Bisimilarity**: Union of all bisimulations

More Potential Insights

- Specification Refinements
- Feasible Logics for BIOMICS ASMs (rather counterintuitive)
- Monads and their Adjunctions

Monads

Definition (Monad)

A monad (T, η, μ) in a category $\mathcal{C}$ consists of an endofunctor $T\colon \mathcal{C} \to \mathcal{C}$ with the following two natural transformations

- **unit** $\eta\colon \iota_{\mathcal{C}} \stackrel{.}{\to} T$
- **multiplication** $\mu\colon T^2 \stackrel{.}{\to} T$

for which the following two diagrams commute:

$$\begin{tikzcd} T^3 \arrow[r, "\mu \circ T"] \arrow[d, "T \circ \mu"'] & T^2 \arrow[d, "\mu"] \\ T^2 \arrow[r, "\mu"] & T \end{tikzcd} \qquad \begin{tikzcd} T \arrow[r, "T \circ \eta"] \arrow[dr, equal] & T^2 \arrow[d, "\mu"] & T \arrow[l, "\eta_T"'] \arrow[dl, equal] \\ & T & \end{tikzcd}$$

First Attempt to Model ASMs using Coalgebra

ASM	Coalgebra
Vocabulary: Functions, Relations, Terms	Given implicitly by a functor, e.g. $T(X) = \prod_i (B_i + C_i \times X)^{A_i}$

First Attempt to Model ASMs using Coalgebra

ASM	Coalgebra
Vocabulary: Functions, Relations, Terms	Given implicitly by a functor, e.g. $T(X) = \prod_i (B_i + C_i \times X)^{A_i}$
Base Set $\mathcal{B}$	arbitrary A_i, B_i, C_i, fixed by T

First Attempt to Model ASMs using Coalgebra

ASM	Coalgebra
Vocabulary: Functions, Relations, Terms	Given implicitly by a functor, e.g. $T(X) = \prod_i (B_i + C_i \times X)^{A_i}$
Base Set $\mathcal{B}$	arbitrary A_i, B_i, C_i, fixed by T
Abstract State	Carrier set U and attributes A

First Attempt to Model ASMs using Coalgebra

ASM	Coalgebra
Vocabulary: Functions, Relations, Terms	Given implicitly by a functor, e.g. $T(X) = \prod_i (B_i + C_i \times X)^{A_i}$
Base Set $\mathcal{B}$	arbitrary A_i, B_i, C_i, fixed by T
Abstract State	Carrier set U and attributes A
Interpretation in a state	Coalgebra $c: U \to T(U)$ (incl. $c_i: U \times A_i \to B_i + C_i \times U$)

First Attempt to Model ASMs using Coalgebra

ASM	Coalgebra
Vocabulary: Functions, Relations, Terms	Given implicitly by a functor, e.g. $T(X) = \prod_i (B_i + C_i \times X)^{A_i}$
Base Set $\mathcal{B}$	arbitrary A_i, B_i, C_i, fixed by T
Abstract State	Carrier set U and attributes A
Interpretation in a state	Coalgebra $c: U \to T(U)$ (incl. $c_i: U \times A_i \to B_i + C_i \times U$)
Initial states $\mathcal{I}$	Initial element $u_0 \in U$

First Attempt to Model ASMs using Coalgebra

ASM	Coalgebra
Vocabulary: Functions, Relations, Terms	Given implicitly by a functor, e.g. $T(X) = \prod_i (B_i + C_i \times X)^{A_i}$
Base Set $\mathcal{B}$	arbitrary A_i, B_i, C_i, fixed by T
Abstract State	Carrier set U and attributes A
Interpretation in a state	Coalgebra $c : U \to T(U)$ (incl. $c_i : U \times A_i \to B_i + C_i \times U$)
Initial states $\mathcal{I}$	Initial element $u_0 \in U$
Tansitions τ_Π	Coalgebra $c : U \to T(U)$

First Attempt to Model ASMs using Coalgebra

ASM	Coalgebra
Vocabulary: Functions, Relations, Terms	Given implicitly by a functor, e.g. $T(X) = \prod_i (B_i + C_i \times X)^{A_i}$
Base Set $\mathcal{B}$	arbitrary A_i, B_i, C_i, fixed by T
Abstract State	Carrier set U and attributes A
Interpretation in a state	Coalgebra $c: \ U \to T(U)$ (incl. $c_i: \ U \times A_i \to B_i + C_i \times U$)
Initial states $\mathcal{I}$	Initial element $u_0 \in U$
Tansitions τ_Π	Coalgebra $c : U \to T(U)$
Composition	Monad M_{ASM}

BIOMICS

First Attempt to Model ASMs using Coalgebra

ASM	Coalgebra
Vocabulary: Functions, Relations, Terms	Given implicitly by a functor, e.g. $T(X) = \prod_i (B_i + C_i \times X)^{A_i}$
Base Set $\mathcal{B}$	arbitrary A_i, B_i, C_i, fixed by T
Abstract State	Carrier set U and attributes A
Interpretation in a state	Coalgebra $c: U \to T(U)$ (incl. $c_i: U \times A_i \to B_i + C_i \times U$)
Initial states $\mathcal{I}$	Initial element $u_0 \in U$
Tansitions τ_Π	Coalgebra $c: U \to T(U)$
Composition	Monad M_{ASM}
Seq. of states or $\Delta(\Pi, I), I \in \mathcal{I}$	maps of c_i to terminal coalgebra

BIOMICS

First Attempt to Model ASMs using Coalgebra

ASM	Coalgebra
Vocabulary: Functions, Relations, Terms	Given implicitly by a functor, e.g. $T(X) = \prod_i (B_i + C_i \times X)^{A_i}$
Base Set $\mathcal{B}$	arbitrary A_i, B_i, C_i, fixed by T
Abstract State	Carrier set U and attributes A
Interpretation in a state	Coalgebra $c : U \to T(U)$ (incl. $c_i : U \times A_i \to B_i + C_i \times U$)
Initial states $\mathcal{I}$	Initial element $u_0 \in U$
Tansitions τ_Π	Coalgebra $c : U \to T(U)$
Composition	Monad M_{ASM}
Seq. of states or $\Delta(\Pi, I), I \in \mathcal{I}$	maps of c_i to terminal coalgebra
Equivalence	Bisimilarity

BIOMICS

Next Steps

- Precise coalgebraic definition
- Terminal coalgebra for ASMs used in BIOMICS
- Find Feasible ASM Monad
- Kleisli and/or Eilenberg Moore categories for ASMs?
- Bialgebras for ASM Modelling?
- Modelling interaction of ASMs with the environment

Conclusions

- Mapping ASMs to coalgebras appear to be *obvious*
- Expressiveness of ASMs complicate coalgebraic definition
- New insights may not be new but have coalgebraic grounding
- Algebraic/coalgebraic basis can become essential for BIOMICS
- BIOMICS may invert the intended use of ASMs (from *synthesis* to *analysis*)
  - Derive BIOMICS interaction machine from ASMs through (co)algebraic refinement
  - Express bio-chemical pathways using ASMs