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)
(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
  Research Agenda in BIOMICS
  Motivation Behind this Paper

ASMs vs. Coalgebraic Specifications
  Abstract State Machine Specifications
  Coalgebras and their Specifications

Potential Insights for ASMs
  From a Coalgebraic Perspective
  From a Monadic Perspective

First Coalgebraic and Monadic Perspective on ASMs

Next Steps and Conclusions
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
  - Tarski: 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
- Tarski: 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 \( \mathcal{V} \) 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 \( \mathcal{V} \)
- Boolean operators and equality sign are in \( \mathcal{V} \)
Abstract States — A Definition

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

- Let $T(\Upsilon)$ 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 $f(t_1, \ldots, t_k) \in T(\Upsilon)$. 
Abstract States — A Definition

- Let $\gamma$ 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 $\gamma$
  - Boolean operators and equality sign are in $\gamma$

- Let $T(\gamma)$ denote a set of **Terms**
  - if $(f, 0) \in \gamma$, then $f \in T(\gamma)$
  - if $(f, k) \in \gamma, t_1, \ldots, t_k \in T(\gamma)$, then $f(t_1, \ldots, t_k) \in T(\gamma)$.

- Let $S(\gamma)$ 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 $\gamma$
Abstract States — A Definition

- Let $\Upsilon$ 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 $\Upsilon$
  - Boolean operators and equality sign are in $\Upsilon$

- Let $T(\Upsilon)$ 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 $f(t_1, \ldots, t_k) \in T(\Upsilon)$.

- Let $S(\Upsilon)$ 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 $\Upsilon$

- **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 $\bar{a}$ has elements from base set $X$, $(f, \bar{a})$ is a **location**
  - let $(f, \bar{a}, b)$ denote an **update** of location $\bar{a}$ with $b$
  - change in memory $\Delta$ by update $u$ is denoted by $\Delta(u, S)$
Updates Transit Between Abstract States

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

- **Update rules**
  - terms as arguments allow for programming with updates
  - update rules $R$ of vocabulary $\mathcal{Y}$ have the form
    $$f(t_1, \ldots, t_k) := t_0$$
    with $k$-ary function $f$ and terms $t_0, \ldots, t_k$
  - 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 $\Delta$ by update $u$ is denoted by $\Delta(u, S)$

- **Update rules**
  - terms as arguments allow for programming with updates
  - update rules $R$ of vocabulary $\Upsilon$ have the form $f(t_1, \ldots, t_k) := t_0$ with $k$-ary function $f$ and terms $t_0, \ldots, t_k$
  - 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 $R_1, \ldots, R_j$
  - change of memory in $S$: $\Delta(R, S) = \Delta(R_1, S) \cup \ldots \cup \Delta(R_j, S)$
Conditional Rules and Programs for ASMs

- Conditional rule $R$
  - Boolean term $\phi$ over $\Upsilon$
  - Rules $R_1, R_2$ in $\Upsilon$
  
    \[
    \text{if } \phi \text{ then } R_1 \text{ else } R_2 \text{ endif}
    \]
  - Common semantics, i.e. $\Delta(R, S) = \Delta(R_1, S)$ if $\phi$ valuates to true in $S$, $\Delta(R_2, S)$ otherwise
Conditional Rules and Programs for ASMs

- Conditional rule $R$
  - Boolean term $\phi$ over $\gamma$
  - Rules $R_1, R_2$ in $\gamma$
    
    \[
    \text{if } \phi \text{ then } R_1 \text{ else } R_2 \text{ endif}
    \]
  - Common semantics, i.e. $\Delta(R, S) = \Delta(R_1, S)$ if $\phi$ valuates to true in $S$, $\Delta(R_2, S)$ otherwise

- Program $\Pi$ over $\gamma$
  - $\Pi$ is a rule over $\gamma$
  - $\Delta(\Pi, S)$ is well defined for every state $S$ defined by $\gamma$
  - changes applied to $S$ by $\Pi$ are defined by $\tau_\Pi(S) = S + \Delta(\Pi, S)$
A Small Example for an ASM Rule/Program

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

\[
\begin{align*}
\text{if } & \quad b = 0 \quad \text{then } \quad d := a \\
\text{else if } & \quad b = 1 \quad \text{then } \quad d := 1 \\
\text{else} & \\
& \quad \text{par} \\
& \quad \quad a := b \\
& \quad \quad b := a \mod b \\
& \quad \text{endpar} \\
\text{endif}
\end{align*}
\]

\((a = 12, b = 6, d = \bot) s_1\)
A Small Example for an ASM Rule/Program

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

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

$(a = 12, b = 6, d = \bot)s_1 \to (6, 0, \bot)s_2$
A Small Example for an ASM Rule/Program

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

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

$(a = 12, b = 6, d = \bot)S_1 \rightarrow (6, 0, \bot)S_2 \rightarrow (6, 0, 6)S_3$
Abstract State Machine (ASM)

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

- Vocabulary $\Upsilon$
- Program $\Pi$ over $\Upsilon$
- Base set $\mathcal{B}$
- Set of abstract states $\mathcal{S}(\Pi)$
- Set of initial states $\mathcal{I}(\Pi) \subset \mathcal{S}(\Pi)$
- transition function $\tau_{\Pi}$
Coalgebraic & Monadic Perspectives on ASMs

Coalgebras — A Definition

Definition (F-Coalgebras)

Given are a category \( \mathcal{C} = (O, M, s, t, c) \) and functor \( F : \mathcal{C} \to \mathcal{C} \). \( \mathcal{C} = \langle C, \tau \rangle_F \) is called \( F \)-coalgebra, if \( C \in O \), and \( \tau \in M \) with \( \tau : C \to F(C) \). \( C \) is called carrier ‘set’ (or state space) and \( \tau \) is called structure (or transition) of coalgebra \( \mathcal{C} \).
Coalgebras Structure Codomains

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

\[ i := 5 \quad \quad S \xrightarrow{i:=5} S \]
Coalgebraic & Monadic Perspectives on ASMs

ASMs vs. Coalgebraic Specifications

Coalgebras and their Specifications

Coalgebras Structure Codomains

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

  \[ i := 5 \quad S \xrightarrow{i:=5} S \]

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

  \[
  \begin{align*}
  & \text{statement} \\
  S_\perp \xrightarrow{\text{statement}} S_\perp
  \end{align*}
  \]
Coalgebraic & Monadic Perspectives on ASMs

ASMs vs. Coalgebraic Specifications
Coalgebras and their Specifications

Coalgebras Structure Codomains

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

\[ i := 5 \quad \quad S \xrightarrow{\text{\texttt{i:=5}}} S \]

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

\[
\begin{array}{c}
\text{statement} \\
S_\bot \xrightarrow{\text{statement}} S_\bot
\end{array}
\]

- Account for exceptions \( E \)

\[
S \cup \{ \bot \} \cup (S \times E) \xrightarrow{\text{statement}} S \cup \{ \bot \} \cup (S \times E)
\]
Imperative programs without side-effects, exceptions, non-termination, etc.

\[ i := 5 \quad S \xrightarrow{i:=5} S \]

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

\[ \text{statement} \quad S_{\perp} \xrightarrow{\text{statement}} S_{\perp} \]

Account for exceptions \( E \)

\[ S \cup \{ \perp \} \cup (S \times E) \xrightarrow{\text{statement}} S \cup \{ \perp \} \cup (S \times E) \]

Question

What if we kept a constant state set and structure the codomain?
Coalgebras are about Observations

- Consider sequences $A^\infty$ 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 $\text{next}$

  \[
  \text{next} : A^\infty \longrightarrow \{\bot\} \cup (A \times A^\infty)
  \]

  defined by the total function $\sigma$ with symbol $\bot$

  \[
  \sigma \longmapsto \begin{cases} \bot & \text{if } \sigma \text{ is the empty sequence} \\ (a, \sigma') & \text{if } \sigma = a \cdot \sigma' \text{ with head } a \text{ and tail } \sigma' \end{cases}
  \]

- Applying $\text{next}$ generates all observable elements of $A^\infty$
Coalgebraic & Monadic Perspectives on ASMs

Coalgebras and Behaviours

- Properties of the function \textit{next}
  - coalgebra of type $\{\bot\} \cup (A \times (-))$
  - terminal (final) coalgebra among all $F$-coalgebras with $F(S) = \{\bot\} \cup (A \times S)$
Coalgebras and Behaviours

- Properties of the function $\text{next}$
  - 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 $\text{behaviour}$ function $\text{beh}_c : S \rightarrow A^\infty$ for any $F$-coalgebra with state space $S$ and $c : S \rightarrow \{\bot\} \cup (A \times S)$
Coalgebraic & Monadic Perspectives on ASMs

ASMs vs. Coalgebraic Specifications

Coalgebras and their Specifications

Coalgebras and Behaviours

- Properties of the function \textit{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 \textit{behaviour} function \(beh_c : S \rightarrow A^\infty\) for any F-coalgebra with state space \(S\) and \(c : S \rightarrow \{\bot\} \cup (A \times S)\)
  - \(beh_c\) is a homomorphism of coalgebras

\[
\begin{array}{c}
\{\bot\} \cup (A \times S) \\
\text{\textup{\textup{\textup{\textit{\textup{id}}}} \cup \text{\textup{\textup{\textit{\textup{id}}}} \times \text{\textup{\textup{\textit{\textup{beh}}}}}}}} \quad \downarrow
\end{array}
\quad \begin{array}{c}
\bot \cup (A \times A^\infty) \\
\text{\textup{\textup{\textit{\textup{next}}}}} \quad \downarrow
\end{array}
\begin{array}{c}
S \\
\text{\textup{\textup{\textit{\textup{beh}}}}} \quad \downarrow
\end{array}
\begin{array}{c}
A^\infty
\end{array}
\]
Terminal $F$-coalgebra determines behaviour for any coalgebra in category $\text{CoAlg}(F)$.
Terminal $F$-coalgebras and their Existence

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

\[
F(X) \xrightarrow{F(\mathit{beh}_c)} F(Z)
\]

\[
X \xrightarrow{\mathit{beh}_c} Z
\]

- Unfortunately: No guarantee that terminal $F$-coalgebra exists
Terminal $F$-coalgebras and their Existence

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

$$F(X) \xrightarrow{F(beh_c)} F(Z)$$

$\xrightarrow{c}$ $\uparrow$ $\uparrow^\zeta$ $\xrightarrow{beh_c}$ $\xrightarrow{}$ $X$ $\xrightarrow{}$ $Z$

- 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 \textit{simple}
What if the terminal coalgebra does not exist or is hard to find?
Bisimulation is a means for comparing \textit{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**
  - $F : \text{Set} \rightarrow \text{Set}$ be a polynomial functor
  - $X, Y$ arbitrary objects in $\text{Set}$
  - 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**
  - Let $F : \text{Set} \to \text{Set}$ be a polynomial functor.
  - Let $X, Y$ be arbitrary objects in $\text{Set}$.
  - Relation lift sends $R \subseteq X \times Y$ to $\text{Rel}(F)(R) \subseteq F(X) \times F(Y)$.

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

- **Bisimulation**
  - Let $c : X \to F(X)$ and $d : Y \to F(Y)$ be two coalgebras on $F$.
  - Relation $R \subseteq X \times Y$ describes the bisimulation for $c$ and $d$ which is closed under $c$ and $d$:
    \[ \forall x \in X, y \in Y, (x, y) \in R \Rightarrow (c(x), d(y)) \in \text{Rel}(F)(R) \]

- **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 \((T, \eta, \mu)\) in a category \(C\) consists of an endofunctor \(T : C \to C\) with the following two natural transformations

- **unit** \(\eta : 1_C \to T\)
- **multiplication** \(\mu : T^2 \to T\)

for which the following two diagrams commute:
First Attempt to Model ASMs using Coalgebra

<table>
<thead>
<tr>
<th>ASM</th>
<th>Coalgebra</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vocabulary: Functions, Relations, Terms</td>
<td>Given implicitly by a functor, e.g. $T(X) = \prod_i (B_i + C_i \times X)^{A_i}$</td>
</tr>
</tbody>
</table>
First Attempt to Model ASMs using Coalgebra

<table>
<thead>
<tr>
<th>ASM</th>
<th>Coalgebra</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vocabulary: Functions, Rela-</td>
<td>Given implicitly by a functor, e.g. $T(X) = \prod_i (B_i + C_i \times X)^{A_i}$</td>
</tr>
<tr>
<td>tions, Terms</td>
<td>arbitrary $A_i, B_i, C_i$, fixed by $T$</td>
</tr>
<tr>
<td>Base Set $\mathcal{B}$</td>
<td></td>
</tr>
</tbody>
</table>
First Attempt to Model ASMs using Coalgebra

<table>
<thead>
<tr>
<th>ASM</th>
<th>Coalgebra</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vocabulary: Functions, Relations, Terms</td>
<td>Given implicitly by a functor, e.g. $T(X) = \prod_i(B_i + C_i \times X)^{A_i}$</td>
</tr>
<tr>
<td>Base Set $\mathcal{B}$</td>
<td>arbitrary $A_i, B_i, C_i$, fixed by $T$</td>
</tr>
<tr>
<td>Abstract State</td>
<td>Carrier set $U$ and attributes $A$</td>
</tr>
</tbody>
</table>
First Attempt to Model ASMs using Coalgebra

<table>
<thead>
<tr>
<th>ASM</th>
<th>Coalgebra</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vocabulary: Functions, Rela-</td>
<td>Given implicitly by a functor, e.g. $T(X) = \prod_i (B_i + C_i \times X)^{A_i}$</td>
</tr>
<tr>
<td>tions, Terms</td>
<td></td>
</tr>
<tr>
<td>Base Set $\mathcal{B}$</td>
<td>arbitrary $A_i, B_i, C_i$, fixed by $T$</td>
</tr>
<tr>
<td>Abstract State</td>
<td>Carrier set $U$ and attributes $A$</td>
</tr>
<tr>
<td>Interpretation in a state</td>
<td>Coalgebra $c : U \rightarrow T(U)$ (incl. $c_i : U \times A_i \rightarrow B_i + C_i \times U$)</td>
</tr>
</tbody>
</table>
First Attempt to Model ASMs using Coalgebra

<table>
<thead>
<tr>
<th>ASM</th>
<th>Coalgebra</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vocabulary: Functions, Relations, Terms</td>
<td>Given implicitly by a functor, e.g. $T(X) = \prod_i (B_i + C_i \times X)^{A_i}$</td>
</tr>
<tr>
<td>Base Set $B$</td>
<td>arbitrary $A_i$, $B_i$, $C_i$, fixed by $T$</td>
</tr>
<tr>
<td>Abstract State</td>
<td>Carrier set $U$ and attributes $A$</td>
</tr>
<tr>
<td>Interpretation in a state</td>
<td>Coalgebra $c : U \rightarrow T(U)$ (incl. $c_i : U \times A_i \rightarrow B_i + C_i \times U)$</td>
</tr>
<tr>
<td>Initial states $\mathcal{I}$</td>
<td>Initial element $u_0 \in U$</td>
</tr>
</tbody>
</table>
First Attempt to Model ASMs using Coalgebra

<table>
<thead>
<tr>
<th>ASM</th>
<th>Coalgebra</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vocabulary: Functions, Relations, Terms</td>
<td>Given implicitly by a functor, e.g. $T(X) = \prod_i (B_i + C_i \times X)^{A_i}$</td>
</tr>
<tr>
<td>Base Set $B$</td>
<td>arbitrary $A_i, B_i, C_i$, fixed by $T$</td>
</tr>
<tr>
<td>Abstract State</td>
<td>Carrier set $U$ and attributes $A$</td>
</tr>
<tr>
<td>Interpretation in a state</td>
<td>Coalgebra $c : U \rightarrow T(U)$ (incl. $c_i : U \times A_i \rightarrow B_i + C_i \times U$)</td>
</tr>
<tr>
<td>Initial states $\mathcal{I}$</td>
<td>Initial element $u_0 \in U$</td>
</tr>
<tr>
<td>Tansitions $\tau_P$</td>
<td>Coalgebra $c : U \rightarrow T(U)$</td>
</tr>
</tbody>
</table>
First Attempt to Model ASMs using Coalgebra

<table>
<thead>
<tr>
<th>ASM</th>
<th>Coalgebra</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Vocabulary:</strong> Functions, Relations, Terms</td>
<td>Given implicitly by a functor, e.g. $T(X) = \prod_i (B_i + C_i \times X)^{A_i}$</td>
</tr>
<tr>
<td><strong>Base Set $B$</strong></td>
<td>arbitrary $A_i$, $B_i$, $C_i$, fixed by $T$</td>
</tr>
<tr>
<td><strong>Abstract State</strong></td>
<td>Carrier set $U$ and attributes $A$</td>
</tr>
<tr>
<td><strong>Interpretation in a state</strong></td>
<td>Coalgebra $c : U \rightarrow T(U)$ (incl. $c_i : U \times A_i \rightarrow B_i + C_i \times U$)</td>
</tr>
<tr>
<td><strong>Initial states $I$</strong></td>
<td>Initial element $u_0 \in U$</td>
</tr>
<tr>
<td><strong>Tansitions $\tau_I$</strong></td>
<td>Coalgebra $c : U \rightarrow T(U)$</td>
</tr>
<tr>
<td><strong>Composition</strong></td>
<td>Monad $M_{ASM}$</td>
</tr>
</tbody>
</table>
### First Attempt to Model ASMs using Coalgebra

<table>
<thead>
<tr>
<th>ASM</th>
<th>Coalgebra</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Vocabulary:</strong> Functions, Relations, Terms</td>
<td>Given implicitly by a functor, e.g. $T(X) = \prod_i (B_i + C_i \times X)^{A_i}$</td>
</tr>
<tr>
<td><strong>Base Set $\mathcal{B}$</strong></td>
<td>arbitrary $A_i$, $B_i$, $C_i$, fixed by $T$</td>
</tr>
<tr>
<td><strong>Abstract State</strong></td>
<td>Carrier set $U$ and attributes $A$</td>
</tr>
<tr>
<td><strong>Interpretation in a state</strong></td>
<td>Coalgebra $c : U \rightarrow T(U)$ (incl. $c_i : U \times A_i \rightarrow B_i + C_i \times U$)</td>
</tr>
<tr>
<td><strong>Initial states $\mathcal{I}$</strong></td>
<td>Initial element $u_0 \in U$</td>
</tr>
<tr>
<td><strong>Tansitions $\tau_{\Pi}$</strong></td>
<td>Coalgebra $c : U \rightarrow T(U)$</td>
</tr>
<tr>
<td><strong>Composition</strong></td>
<td>Monad $M_{ASM}$</td>
</tr>
<tr>
<td><strong>Seq. of states or $\Delta(\Pi, I), I \in \mathcal{I}$</strong></td>
<td>maps of $c_i$ to terminal coalgebra</td>
</tr>
</tbody>
</table>
First Attempt to Model ASMs using Coalgebra

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