A decision procedure for bisimilarity of generalized regular expressions

M. Bonsangue[1,2] G. Caltais[5] E. Goriac[5] D. Lucanu[4]
J. Rutten[1,3] A. Silva[1]

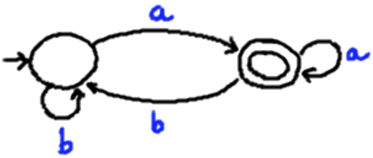
[1]Centrum Wiskunde & Informatica, The Netherlands
[2]LIACS - Leiden University, The Netherlands
[3]Radboud Universiteit Nijmegen, The Netherlands
[4]Faculty of Computer Science - Alexandru Ioan Cuza University, Romania
[5]School of Computer Science - Reykjavik University, Iceland

SBMF'10, November 2010

Motivation

Deterministic automata (DA)

- Widely used model in Computer Science.
- Acceptors of languages



Regular expressions

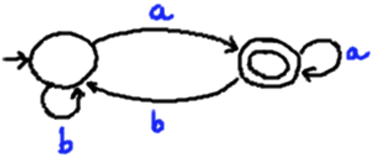
- *User-friendly* alternative to DA notation.
- Many applications: pattern matching (`grep`), specification of circuits, ...

$$b^*a(b^*a)^*$$

Motivation

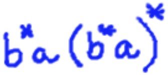
Deterministic automata (DA)

- Widely used model in Computer Science.
- Acceptors of languages



Regular expressions

- *User-friendly* alternative to DA notation.
- Many applications: pattern matching (`grep`), specification of circuits, ...

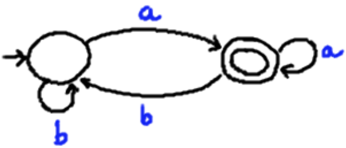


Kleene's Theorem

Let $A \subseteq \Sigma^*$. The following are equivalent.

1. $A = L(\mathcal{A})$, for some finite automaton $\mathcal{A}$.
2. $A = L(r)$, for some regular expression r.

Motivation



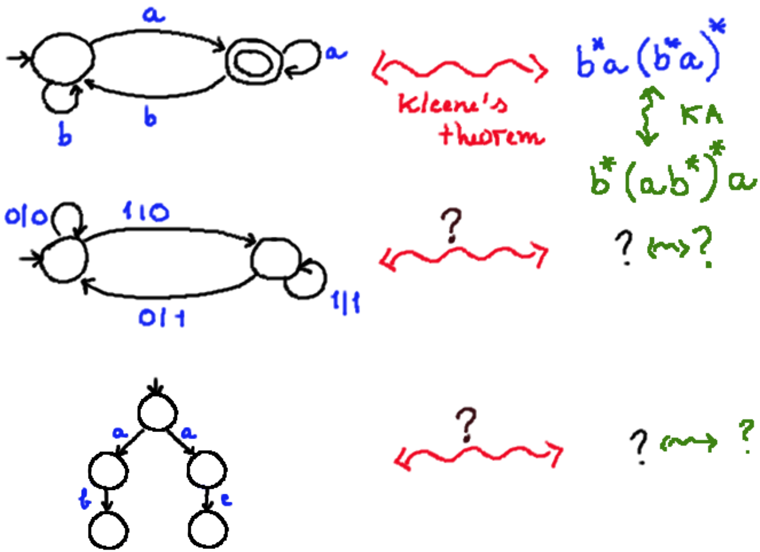
Kleene's theorem

$b^*a(b^*a)^*$

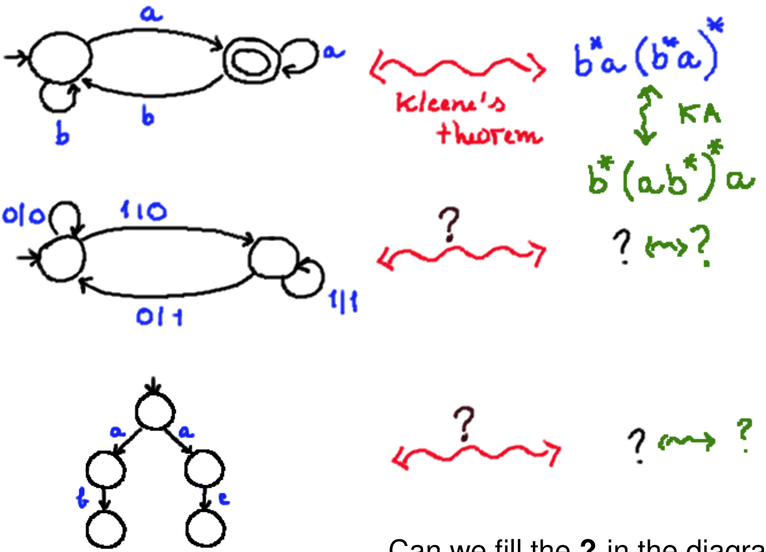
KA

$b^*(ab^*)^*a$

Motivation



Motivation



Can we fill the **?** in the diagram?

In previous work . . .

We presented:

- a generalized notion of regular expressions;
- an analogue of Kleene's theorem;
- and sound and complete axiomatizations with respect to bisimilarity

for a large class of systems (labelled transition systems, Mealy machines, probabilistic automata).

All the above was derived modularly from the type of each system.

Question: Can we automate the reasoning on equivalence of expressions, also in a modular way?

In previous work . . .

We presented:

- a generalized notion of regular expressions;
- an analogue of Kleene's theorem;
- and sound and complete axiomatizations with respect to bisimilarity

for a large class of systems (labelled transition systems, Mealy machines, probabilistic automata).

All the above was derived modularly from the type of each system.

Question: Can we automate the reasoning on equivalence of expressions, also in a modular way?

In previous work . . .

We presented:

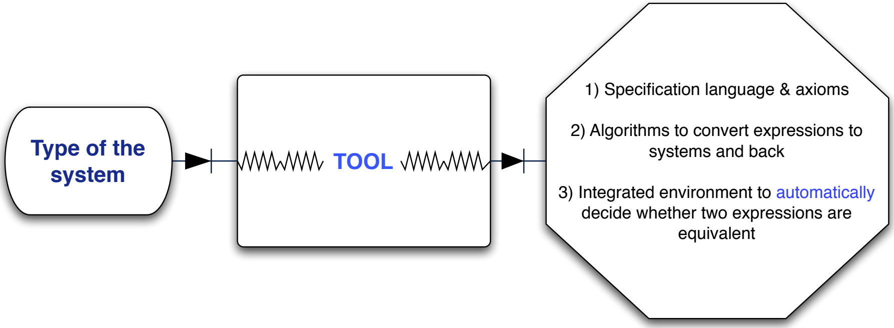
- a generalized notion of regular expressions;
- an analogue of Kleene's theorem;
- and sound and complete axiomatizations with respect to bisimilarity

for a large class of systems (labelled transition systems, Mealy machines, probabilistic automata).

All the above was derived modularly from the type of each system.

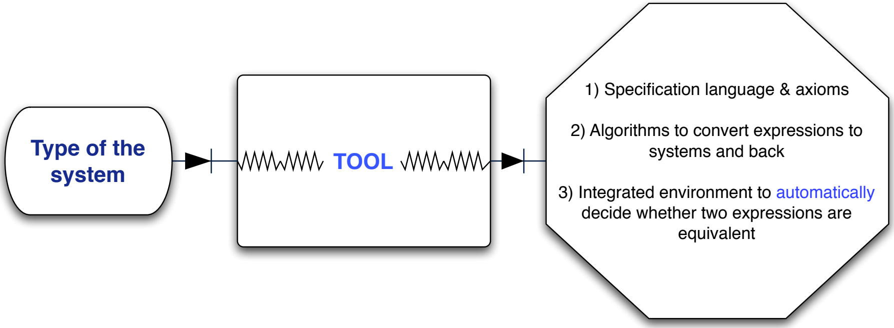
Question: Can we automate the reasoning on equivalence of expressions, also in a modular way?

The ultimate goal. . .



In this talk, we will be focusing on 1) and 3).

The ultimate goal...

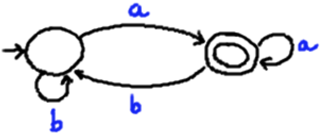


In this talk, we will be focusing on 1) and 3).

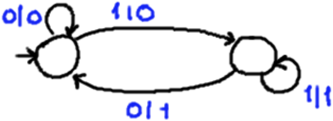
Outline

- Generalized regular expressions
- Equivalence of expressions
- Snapshot of the tool

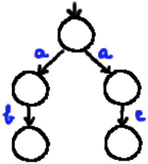
What do these things have in common?



$(S, \delta : S \to 2 \times S^A)$

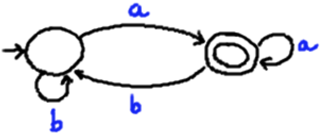


$(S, \delta : S \to (B \times S)^A)$

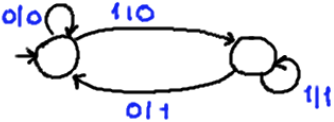


$(S, \delta : S \to 1 + (\mathcal{P}S)^A)$

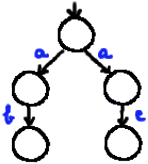
What do these things have in common?



$$(S, \delta : S \to 2 \times S^A)$$

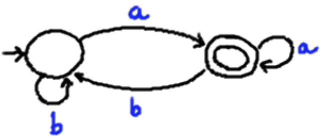


$$(S, \delta : S \to (B \times S)^A)$$

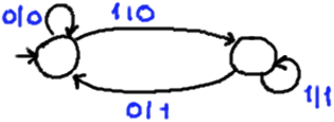


$$(S, \delta : S \to 1 + (\mathcal{P}S)^A)$$

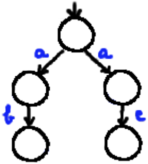
What do these things have in common?



$$(S, \delta : S \to 2 \times S^A)$$

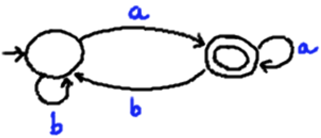


$$(S, \delta : S \to (B \times S)^A)$$

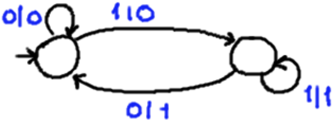


$$(S, \delta : S \to 1 + (\mathcal{P}S)^A)$$

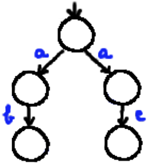
What do these things have in common?



$$(S, \delta : S \to 2 \times S^A)$$

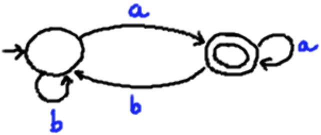


$$(S, \delta : S \to (B \times S)^A)$$

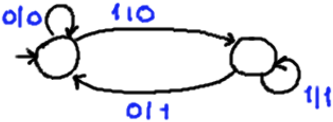


$$(S, \delta : S \to 1 + (\mathcal{P}S)^A)$$

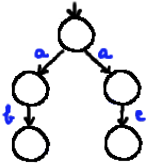
What do these things have in common?



$(S, \delta : S \to 2 \times S^A)$

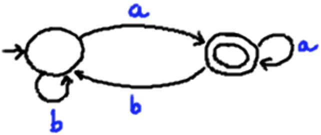


$(S, \delta : S \to (B \times S)^A)$

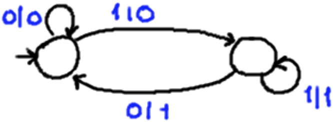


$(S, \delta : S \to 1 + (\mathcal{P}S)^A)$

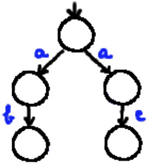
What do these things have in common?



$$(S, \delta : S \to 2 \times S^A)$$



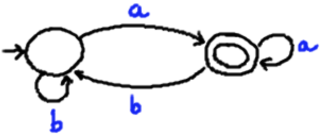
$$(S, \delta : S \to (B \times S)^A)$$



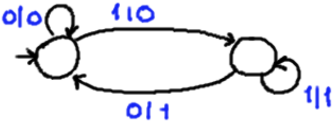
$$(S, \delta : S \to 1 + (\mathcal{P}S)^A)$$

$$(S, \delta : S \to \mathcal{G}S)$$

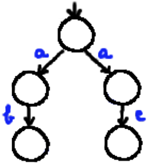
What do these things have in common?



$(S, \delta : S \to 2 \times S^A)$



$(S, \delta : S \to (B \times S)^A)$



$(S, \delta : S \to 1 + (\mathcal{P}S)^A)$

$(S, \delta : S \to \mathcal{G}S)$ $\mathcal{G}$**-coalgebras**

Coalgebras

- Generalizations of deterministic automata
- Set of states S and a transition function $t : S \to \mathcal{G}S$ where $\mathcal{G}$ encodes the type of the system:

$$\mathcal{G} ::= Id \mid B \mid \mathcal{G} \times \mathcal{G} \mid \mathcal{G} + \mathcal{G} \mid \mathcal{G}^A \mid \mathcal{P}\mathcal{G}$$

$\mathcal{P}$ finite

Examples

- $\mathcal{G} = 2 \times Id^A$ Deterministic automata
- $\mathcal{G} = (B \times Id)^A$ Mealy machines
- $\mathcal{G} = 1 + (\mathcal{P}Id)^A$ LTS (with explicit termination)
- ...

Coalgebras

- Generalizations of deterministic automata
- Set of states S and a transition function $t : S \to \mathcal{G}S$ where $\mathcal{G}$ encodes the type of the system:

$$\mathcal{G} ::= Id \mid B \mid \mathcal{G} \times \mathcal{G} \mid \mathcal{G} + \mathcal{G} \mid \mathcal{G}^A \mid \mathcal{P}\mathcal{G}$$

$\mathcal{P}$ finite

Examples

- $\mathcal{G} = 2 \times Id^A$ Deterministic automata
- $\mathcal{G} = (B \times Id)^A$ Mealy machines
- $\mathcal{G} = 1 + (\mathcal{P}Id)^A$ LTS (with explicit termination)
- ...

The power of $\mathcal{G}$

The functor $\mathcal{G}$ determines:

1. notion of observational equivalence (coalg. bisimulation)
2. behaviour (final coalgebra)
3. set of expressions describing finite systems
4. axioms to prove bisimulation equivalence of expressions

The power of $\mathcal{G}$

The functor $\mathcal{G}$ determines:

1. notion of observational equivalence (coalg. bisimulation)
2. behaviour (final coalgebra)
3. set of expressions describing finite systems
4. axioms to prove bisimulation equivalence of expressions

The power of $\mathcal{G}$

The functor $\mathcal{G}$ determines:

1. notion of observational equivalence (coalg. bisimulation)
2. behaviour (final coalgebra)
3. set of expressions describing finite systems
4. axioms to prove bisimulation equivalence of expressions

1 + 2 are standard universal coalgebra; 3 + 4 are [BRS10]

In a nutshell — beyond deterministic automata

Deterministic automata $Q \to 2 \times Q^{\Sigma}$	$\rightsquigarrow$	$\mathcal{G}$-coalgebras $Q \to \mathcal{G}Q$
$\Updownarrow$		$\Updownarrow$
Regular Expressions	$\rightsquigarrow$	$\mathcal{G}$-expressions
$\Updownarrow$		$\Updownarrow$
Formal Languages	$\rightsquigarrow$	Final coalgebra

$\mathcal{G}$-expressions

$$E \quad ::= \quad \underline{\emptyset} \mid \epsilon \mid E \cdot E \mid E + E \mid E^*$$

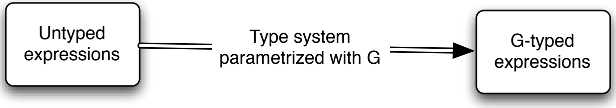
$$E_{\mathcal{G}} \quad ::= \quad ?$$

$\mathcal{G}$-expressions

$$E \quad ::= \quad \underline{\emptyset} \mid \epsilon \mid E \cdot E \mid E + E \mid E^*$$

$$E_{\mathcal{G}} \quad ::= \quad ?$$

How do we define $E_{\mathcal{G}}$?



Examples

Deterministic automata expressions – $\mathcal{G} = 2 \times Id^A$

$$\varepsilon \quad ::= \quad \underbrace{\underline{\emptyset} \mid \varepsilon \oplus \varepsilon \mid \mu x.\gamma}_{\mathcal{G}} \mid$$

Examples

Deterministic automata expressions – $\mathcal{G} = 2 \times Id^A$

$$\varepsilon \quad ::= \quad \underbrace{\underline{\emptyset} \mid \varepsilon \oplus \varepsilon \mid \mu x.\gamma}_{\mathcal{G}} \mid \qquad\qquad \mid \qquad\qquad$$

$$\underbrace{\qquad\qquad\qquad\qquad\qquad}_{\times}$$

Examples

Deterministic automata expressions – $\mathcal{G} = 2 \times Id^A$

$$\varepsilon \quad ::= \quad \underbrace{\underline{\emptyset} \mid \varepsilon \oplus \varepsilon \mid \mu x.\gamma}_{\mathcal{G}} \mid \underbrace{\underbrace{1}_{2} \mid \underbrace{0}_{2} \mid \underbrace{a(\varepsilon)}_{Id^A}}_{\times}$$

Examples

Deterministic automata expressions – $\mathcal{G} = 2 \times Id^A$

$$\varepsilon \quad ::= \quad \underbrace{\underline{\emptyset} \mid \varepsilon \oplus \varepsilon \mid \mu x.\gamma}_{\mathcal{G}} \mid \underbrace{\underbrace{1}_{2} \mid \underbrace{0}_{2} \mid \underbrace{a(\varepsilon)}_{Id^A}}_{\times}$$

LTS expressions – $\mathcal{G} = 1 + (\mathcal{P} Id)^A$

$$\varepsilon \quad ::= \quad \underline{\emptyset} \mid \varepsilon \oplus \varepsilon \mid \mu x.\gamma$$

Examples

Deterministic automata expressions – $\mathcal{G} = 2 \times Id^A$

$$\varepsilon \quad ::= \quad \underbrace{\underline{\emptyset} \mid \varepsilon \oplus \varepsilon \mid \mu x.\gamma}_{\mathcal{G}} \mid \underbrace{\underbrace{1}_{2} \mid \underbrace{0}_{2} \mid \underbrace{a(\varepsilon)}_{Id^A}}_{\times}$$

LTS expressions – $\mathcal{G} = 1 + (\mathcal{P}Id)^A$

$$\varepsilon \quad ::= \quad \underline{\emptyset} \mid \varepsilon \oplus \varepsilon \mid \mu x.\gamma \mid \underbrace{\surd}_{1} \mid \underbrace{\partial}_{(\mathcal{P}Id)^A} \mid \underbrace{a.\varepsilon}_{(\mathcal{P}Id)^A}$$

Operational semantics

The set of $\mathcal{G}$-expressions has a coalgebraic structure given by

$$\delta_{\mathcal{G}} : \mathsf{Exp}_{\mathcal{G}} \to \mathcal{G}(\mathsf{Exp}_{\mathcal{G}})$$

$\delta_{\mathcal{G}}$ ...

- ...provides an operational semantics for the set of expressions
- ...defines the dynamics of the system
- ...is used for observing the behaviour of the system

Operational semantics

The set of $\mathcal{G}$-expressions has a coalgebraic structure given by

$$\delta_{\mathcal{G}} : \mathsf{Exp}_{\mathcal{G}} \to \mathcal{G}(\mathsf{Exp}_{\mathcal{G}})$$

$\delta_{\mathcal{G}}$ ...

- ...provides an operational semantics for the set of expressions
- ...defines the dynamics of the system
- ...is used for observing the behaviour of the system

Operational semantics

The set of $\mathcal{G}$-expressions has a coalgebraic structure given by

$$\delta_{\mathcal{G}} : \mathsf{Exp}_{\mathcal{G}} \to \mathcal{G}(\mathsf{Exp}_{\mathcal{G}})$$

$\delta_{\mathcal{G}}$ ...

- ...provides an operational semantics for the set of expressions
- ...defines the dynamics of the system
- ...is used for observing the behaviour of the system

Operational semantics

The set of $\mathcal{G}$-expressions has a coalgebraic structure given by

$$\delta_{\mathcal{G}} : \mathsf{Exp}_{\mathcal{G}} \to \mathcal{G}(\mathsf{Exp}_{\mathcal{G}})$$

$\delta_{\mathcal{G}}$ ...

- ...provides an operational semantics for the set of expressions
- ...defines the dynamics of the system
- ...is used for observing the behaviour of the system

The goal: proving equivalence

- Automatically proving equivalence (bisimilarity) of expressions;
- Tool: `Circ`

- Meta-language implemented in `Maude`
- **Input**: Algebraic specification + coalgebraic structure (dynamics)
- **Engine**: circular coinduction – constructing bisimulation
- **Output**: Yes / No / **?**

The goal: proving equivalence

- Automatically proving equivalence (bisimilarity) of expressions;
- Tool: `Circ`

- Meta-language implemented in `Maude`
- **Input**: Algebraic specification + coalgebraic structure (dynamics)
- **Engine**: circular coinduction – constructing bisimulation
- **Output**: Yes / No / **?**

The goal: proving equivalence

- Automatically proving equivalence (bisimilarity) of expressions;
- Tool: `Circ`

- Meta-language implemented in `Maude`
- **Input**: Algebraic specification + coalgebraic structure (dynamics)
- **Engine**: circular coinduction – constructing bisimulation
- **Output**: Yes / No / **?**

Our challenge

$$
\begin{tikzcd}
\mathrm{Exp}_{\mathcal{G}} \arrow[dr, Rightarrow] & & \\
\text{coalgebraic structure}(\delta_{\mathcal{G}}) \arrow[r, Rightarrow] & \text{algebraic specification} \arrow[r] & \mathtt{CIRC} \arrow[dd, Rightarrow, "\varepsilon_1 \sim_{\mathcal{G}} \varepsilon_2"] \\
\mathcal{G} - \text{bisimilarity} \arrow[r, Rightarrow] & \text{deduction relation} \arrow[ur] & \\
 & & \textit{YES}/\textit{NO}
\end{tikzcd}
$$

How to guarantee that the prover always says YES/NO?

Our challenge

$$
\begin{tikzcd}
\mathrm{Exp}_{\mathcal{G}} \arrow[dr, Rightarrow] & & \\
\text{coalgebraic structure}(\delta_{\mathcal{G}}) \arrow[r, Rightarrow] & \text{algebraic specification} \arrow[r] & \mathtt{CIRC} \arrow[dd, Rightarrow, "\varepsilon_1 \sim_{\mathcal{G}} \varepsilon_2"] \\
\mathcal{G} - \text{bisimilarity} \arrow[r, Rightarrow] & \text{deduction relation} \arrow[ur] & \\
 & & \textit{YES} / \textit{NO}
\end{tikzcd}
$$

How to guarantee that the prover always says YES/NO?

Unraveling can be infinite

$$\varepsilon = \mu x.a(b(x)) \oplus 1$$

$$\varepsilon \xrightarrow{\delta_a} \langle 1, b(\varepsilon) \rangle \xrightarrow{\delta_b} \langle 1, \varepsilon \rangle$$

Unraveling can be infinite

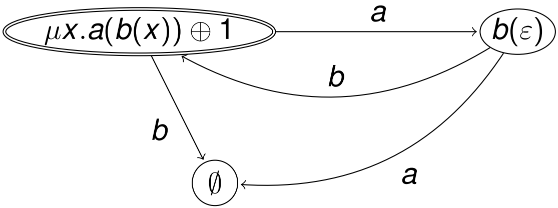
$$\varepsilon = \mu x.a(b(x)) \oplus 1$$

$$\begin{tikzcd}
\varepsilon \arrow[r, "\delta_a"] \arrow[dr, "\delta_b"'] & \langle 1, b(\varepsilon) \rangle \arrow[r, "\delta_b"] \arrow[d, "\delta_a"] & \langle 1, \varepsilon \rangle \\
& \langle 0, \emptyset \rangle &
\end{tikzcd}$$

Unraveling can be infinite

$$\varepsilon = \mu x.a(b(x)) \oplus 1$$

$$\begin{tikzcd}
\varepsilon \arrow[r, "\delta_a"] \arrow[dr, "\delta_b"'] & \langle 1, b(\varepsilon) \rangle \arrow[r, "\delta_b"] \arrow[d, "\delta_a"] & \langle 1, \varepsilon \rangle \\
& \langle 0, \emptyset \rangle &
\end{tikzcd}$$



Unraveling can be infinite

But . . .

$$\varepsilon = \mu x.a(x \oplus x)$$

Unraveling can be infinite

But ...

$$\varepsilon = \mu x.a(x \oplus x)$$

$\varepsilon \overset{\delta}{\longmapsto} \langle 0, \varepsilon \oplus \varepsilon \rangle$

Unraveling can be infinite

But ...

$$\varepsilon = \mu x.a(x \oplus x)$$

$$\varepsilon \xmapsto{\delta} \langle 0, \varepsilon \oplus \varepsilon \rangle \xmapsto{\delta} \langle 0, (\varepsilon \oplus \varepsilon) \oplus (\varepsilon \oplus \varepsilon) \rangle \xmapsto{\delta} \langle 0, (\varepsilon \oplus \varepsilon) \oplus (\varepsilon \oplus \varepsilon) \oplus (\varepsilon \oplus \varepsilon) \rangle \ldots$$

Unraveling can be infinite

But ...

$$\varepsilon = \mu x.a(x \oplus x)$$

$$\varepsilon \xmapsto{\delta} \langle 0, \varepsilon \oplus \varepsilon \rangle \xmapsto{\delta} \langle 0, (\varepsilon \oplus \varepsilon) \oplus (\varepsilon \oplus \varepsilon) \rangle \xmapsto{\delta} \langle 0, (\varepsilon \oplus \varepsilon) \oplus (\varepsilon \oplus \varepsilon) \oplus (\varepsilon \oplus \varepsilon) \rangle \ldots$$

We need **ACI**!

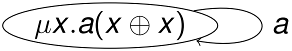
Unraveling can be infinite

But ...

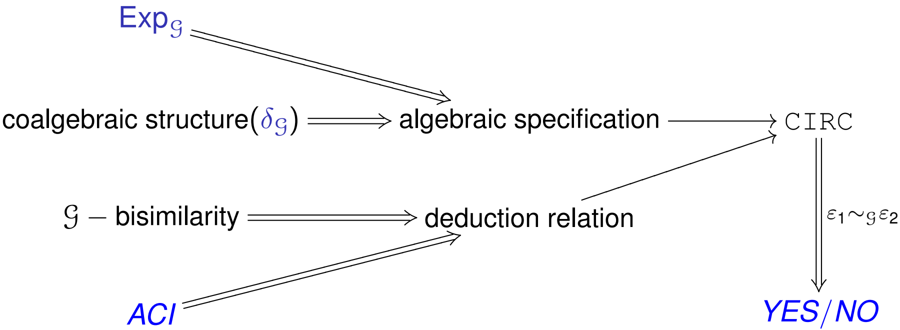
$$\varepsilon = \mu x.a(x \oplus x)$$

$$\varepsilon \overset{\delta}{\longmapsto} \langle 0, \varepsilon \oplus \varepsilon \rangle \overset{\delta}{\longmapsto} \langle 0, (\varepsilon \oplus \varepsilon) \oplus (\varepsilon \oplus \varepsilon) \rangle \overset{\delta}{\longmapsto} \langle 0, (\varepsilon \oplus \varepsilon) \oplus (\varepsilon \oplus \varepsilon) \oplus (\varepsilon \oplus \varepsilon) \rangle \ldots$$

We need **ACI**!



A decision procedure for bisimilarity



- Adding equations is possible because `Circ` is an extension of `Maude`
- With ACI, the *bisimulation game* is decidable!

Algebra meets coalgebra

A decision procedure for bisimilarity

$$\begin{tikzcd}
\mathrm{Exp}_{\mathcal{G}} \arrow[dr, Rightarrow] & & \\
\text{coalgebraic structure}(\delta_{\mathcal{G}}) \arrow[r, Rightarrow] & \text{algebraic specification} \arrow[r] & \mathtt{CIRC} \arrow[dd, Rightarrow, "\varepsilon_1 \sim_{\mathcal{G}} \varepsilon_2"] \\
\mathcal{G} - \text{bisimilarity} \arrow[r, Rightarrow] & \text{deduction relation} \arrow[ur] & \\
\mathit{ACI} \arrow[ur, Rightarrow] & & \mathit{YES}/\mathit{NO}
\end{tikzcd}$$

- Adding equations is possible because `Circ` is an extension of `Maude`
- With ACI, the *bisimulation game* is decidable!

Algebra meets coalgebra

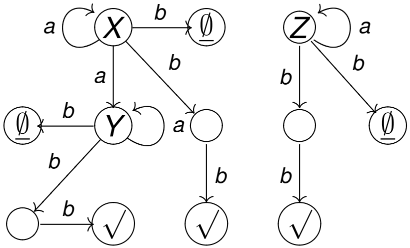
Example

$$
\begin{aligned}
\varepsilon_X &= \mu x.a.x \oplus b.\underline{\emptyset} \oplus a.(\mu y.a.y \oplus b.\underline{\emptyset} \oplus b.b.\surd) \oplus b.b.\surd \\
\varepsilon_Z &= \mu z.a.z \oplus b.\underline{\emptyset} \oplus b.b.\surd
\end{aligned}
$$

Question Are ε_X and ε_Z equivalent?

Example

$$\begin{aligned}\varepsilon_X &= \mu x.a.x \oplus b.\underline{\emptyset} \oplus a.(\mu y.a.y \oplus b.\underline{\emptyset} \oplus b.b.\surd) \oplus b.b.\surd \\ \varepsilon_Z &= \mu z.a.z \oplus b.\underline{\emptyset} \oplus b.b.\surd\end{aligned}$$



Question Are ε_X and ε_Z equivalent?

Tool Snapshot

```
Maude> in protofunctorizer.maude
(alph A is a b endalph)
(alph B is tick endalph)
(functor B + P(Id)^A .)
(E1 = mu X. ... )
(E2 = mu Z. ... )
(set goal E1 = E2 .)
(generate coalgebra .)
296 lines of CIRC specification!
Maude> in circ.maude XandZ.maude
(coinduction .)
```

Proof succeeded.

$$\mathcal{G} = B + \mathcal{P}(Id)^A, A = \{a, b\}, B = \{\surd\}$$

Tool Snapshot

```
Maude> in protofunctorizer.maude
(alph A is a b endalph)
(alph B is tick endalph)
(functor B + P(Id)^A .)
(E1 = mu X. ...  )
(E2 = mu Z. ...  )
(set goal E1 = E2 .)
(generate coalgebra .)
296 lines of CIRC specification!
Maude> in circ.maude XandZ.maude
(coinduction .)
```

Proof succeeded.

$\mathcal{G} = B + \mathcal{P}(Id)^A, A = \{a, b\}, B = \{\surd\}$

Tool Snapshot

```
Maude> in protofunctorizer.maude
(alph A is a b endalph)
(alph B is tick endalph)
(functor B + P(Id)^A .)
(E1 = mu X. ...  )
(E2 = mu Z. ...  )
(set goal E1 = E2 .)
(generate coalgebra .)
296 lines of CIRC specification!
Maude> in circ.maude XandZ.maude
(coinduction .)
```

Proof succeeded.

$\mathcal{G} = B + \mathcal{P}(Id)^A, A = \{a, b\}, B = \{\surd\}$

Tool Snapshot

```
 Maude> in protofunctorizer.maude
(alph A is a b endalph)
(alph B is tick endalph)
(functor B + P(Id)^A .)
(E1 = mu X. ...  )
(E2 = mu Z. ...  )
(set goal E1 = E2 .)
(generate coalgebra .)
296 lines of CIRC specification!
Maude> in circ.maude XandZ.maude
(coinduction .)
```

Proof succeeded.

$$\mathcal{G} = B + \mathcal{P}(Id)^A, A = \{a, b\}, B = \{\surd\}$$

Tool Snapshot

```
Maude> in protofunctorizer.maude
(alph A is a b endalph)
(alph B is tick endalph)
(functor B + P(Id)^A .)
(E1 = mu X. ...  )
(E2 = mu Z. ...  )
(set goal E1 = E2 .)
(generate coalgebra .)
296 lines of CIRC specification!
Maude> in circ.maude XandZ.maude
(coinduction .)
```

Proof succeeded.

$$\mathcal{G} = B + \mathcal{P}(Id)^A, A = \{a, b\}, B = \{\surd\}$$

Tool Snapshot

```
Maude> in protofunctorizer.maude
(alph A is a b endalph)
(alph B is tick endalph)
(functor B + P(Id)^A .)
(E1 = mu X. ...  )
(E2 = mu Z. ...  )
(set goal E1 = E2 .)
(generate coalgebra .)
296 lines of CIRC specification!
Maude> in circ.maude XandZ.maude
(coinduction .)
```

Proof succeeded.

$$\mathcal{G} = B + \mathcal{P}(Id)^A, A = \{a, b\}, B = \{\surd\}$$

Tool Snapshot

```
Maude> in protofunctorizer.maude
(alph A is a b endalph)
(alph B is tick endalph)
(functor B + P(Id)^A .)
(E1 = mu X. ...  )
(E2 = mu Z. ...  )
(set goal E1 = E2 .)
(generate coalgebra .)
296 lines of CIRC specification!
Maude> in circ.maude XandZ.maude
(coinduction .)
```

Proof succeeded.

$$\mathcal{G} = B + \mathcal{P}(Id)^A, A = \{a, b\}, B = \{\surd\}$$

Tool Snapshot

```
Maude> in protofunctorizer.maude
(alph A is a b endalph)
(alph B is tick endalph)
(functor B + P(Id)^A .)
(E1 = mu X. ...  )
(E2 = mu Z. ...  )
(set goal E1 = E2 .)
(generate coalgebra .)
296 lines of CIRC specification!
Maude> in circ.maude XandZ.maude
(coinduction .)
```

Proof succeeded.

$$\mathcal{G} = B + \mathcal{P}(Id)^A, A = \{a, b\}, B = \{\surd\}$$

Tool Snapshot

```
Maude> in protofunctorizer.maude
(alph A is a b endalph)
(alph B is tick endalph)
(functor B + P(Id)^A .)
(E1 = mu X. ...  )
(E2 = mu Z. ...  )
(set goal E1 = E2 .)
(generate coalgebra .)
296 lines of CIRC specification!
Maude> in circ.maude XandZ.maude
(coinduction .)
```

Proof succeeded.

$$\mathcal{G} = B + \mathcal{P}(Id)^A, A = \{a, b\}, B = \{\surd\}$$

Tool Snapshot

```
Maude> in protofunctorizer.maude
(alph A is a b endalph)
(alph B is tick endalph)
(functor B + P(Id)^A .)
(E1 = mu X. ...  )
(E2 = mu Z. ...  )
(set goal E1 = E2 .)
(generate coalgebra .)
296 lines of CIRC specification!
Maude> in circ.maude XandZ.maude
(coinduction .)
```

Proof succeeded.

$$\mathcal{G} = B + \mathcal{P}(Id)^A, A = \{a, b\}, B = \{\surd\}$$

Conclusions and Future work

Conclusions

- Generic framework to uniformly derive language and axioms for a large class of systems
- Generalization of Kleene theorem and Kleene algebra, parametric on the functor.
- Automation in `Circ`: decision procedure for equivalence of expressions.

Future work

- Making the tool more user friendly;
- Extending the class of systems to include quantitative systems (probabilistic automata, etc)
- Apply it to a serious case study (circuit design, compiler optimization, . . .)

Conclusions and Future work

Conclusions

- Generic framework to uniformly derive language and axioms for a large class of systems
- Generalization of Kleene theorem and Kleene algebra, parametric on the functor.
- Automation in `Circ`: decision procedure for equivalence of expressions.

Future work

- Making the tool more user friendly;
- Extending the class of systems to include quantitative systems (probabilistic automata, etc)
- Apply it to a serious case study (circuit design, compiler optimization, . . .)

Thank you!

- for more details see:
  - “Non-deterministic Kleene coalgebras"
    A.Silva, M. Bonsangue, J. Rutten
  - “Circular coinduction - A proof theoretical foundation"
    G. Roşu, D. Lucanu
- QUESTIONS?