Towards Compositional Feedback in Non-Deterministic and Non-Input-Receptive Systems

Stavros Tripakis\textsuperscript{1,2}

Joint LICS’16 paper with Viorel Preoteasa\textsuperscript{2}
Contributions to the RCRS framework also by Iulia Dragomir\textsuperscript{2}

\textsuperscript{1} UC Berkeley, USA
\textsuperscript{2} Aalto University, Finland

ExCAPE PI Meeting 2016
Motivation: compositional reasoning for Simulink

Fuel Control System Model

This model uses only the ODEs to implement the dynamics.

Benchmark provided by Toyota

This is a model of a hybrid automation with polynomial dynamics, and an implementation of the 3rd model that appears in "Powertrain Control Verification Benchmark", 2014 Hybrid Systems: Computation and Control, X. Jin, J. V. Dashmuh, J. Kapinski, K. Ueda, and K. Butts.
What does “compositional reasoning for Simulink” mean?

1. Be able to model basic Simulink blocks: constants, adders, integrators, ...

2. Be able to model arbitrary Simulink models: hierarchical block diagrams.
What does “compositional reasoning for Simulink” mean?

1. Be able to model basic Simulink blocks: constants, adders, integrators, ...

2. Be able to model arbitrary Simulink models: hierarchical block diagrams.

3. Be able to check compatibility: “lightweight verification”, akin to type-checking.

4. Be able to synthesize system from subsystems: compute component compositions bottom-up.

5. Be able to check substitutability: component refinement.
What does “compositional reasoning for Simulink” mean?

1. Be able to model basic Simulink blocks: constants, adders, integrators, ...

2. Be able to model arbitrary Simulink models: **hierarchical block diagrams**.

3. Be able to **check compatibility**: “lightweight verification”, akin to type-checking.

4. Be able to **synthesize system from subsystems**: compute component compositions bottom-up.

5. Be able to **check substitutability**: component **refinement**.

6. Be able to express and verify **safety and liveness** properties.
Relational interfaces: symbolic, synchronous version of Alfaro-Henzinger’s *interface automata*

Example: relational interface for division component

\[
\text{Div} \quad \begin{array}{c} x \\ y \end{array} \rightarrow z \quad \text{contract: } y \neq 0 \land z = \frac{x}{y}
\]

Can model *non-input-receptive* systems:

\[
y \neq 0 \land z = \frac{x}{y} \quad \text{instead of} \quad y \neq 0 \rightarrow z = \frac{x}{y}
\]
Refinement Calculus of Reactive Systems (RCRS)


- Relational interfaces: symbolic, synchronous version of Alfaro-Henzinger’s interface automata
- Example: relational interface for division component

\[
\begin{array}{c}
\xrightarrow{x} \\
\xrightarrow{y} \\
\xrightarrow{z}
\end{array}
\quad \text{Div} \quad \text{contract: } y \neq 0 \land z = \frac{x}{y}
\]

- Can model non-input-receptive systems:

\[
y \neq 0 \land z = \frac{x}{y} \quad \text{instead of} \quad y \neq 0 \rightarrow z = \frac{x}{y}
\]

- RCRS:
  - Extends relational interfaces to handle liveness properties
  - Semantics based on monotonic property transformers
  - Can handle stateful systems, and continuous blocks by Euler discretization, e.g., Integrator: \[ s' = s + x \cdot dt \]
Composition operators

- Serial composition (\(\forall - \exists\) synthesis inside!)

  \[
  x \rightarrow A \quad y \rightarrow B \rightarrow z
  \]

- Parallel composition (conjunction)

  \[
  x \rightarrow A \rightarrow y
  \]
  \[
  z \rightarrow B \rightarrow t
  \]

- Feedback composition

  \[
  x \rightarrow S \rightarrow y
  \]
Composition operators

- Serial composition (∀-∃ synthesis inside!)

- Parallel composition (conjunction)

- Feedback composition

How to define feedback composition?
“Easy” feedback: “broken” by unit delays

\[
a(k) = c(k - 1)
\]

No instantaneous cyclic dependency.
“Easy” feedback: “broken” by unit delays

\[ a(k) = c(k - 1) \]

No instantaneous cyclic dependency.

Handled in our earlier work on relational interfaces:
“Not too hard” feedback

\[ f(u) \]

Output of FuelCmdOpen does not depend on \( u \):

\[
\text{FuelCmdOpen}(u_1, u_2, \ldots, u_7) = 1_7 (-0.366 + 0.08979 u_7 u_3 - 0.0337 u_7 u_32 + 0.0001 u_72 u_3)
\]

No instantaneous cyclic dependency

Handled in recent work:

“Not too hard” feedback

Output of FuelCmdOpen does not depend on $u_6$:

$$\text{FuelCmdOpen}(u_1, u_2, \ldots, u_7) = \frac{1}{14.7}(-0.366 + 0.08979u_7u_3 - 0.0337u_7u_3^2 + 0.0001u_7^2u_3)$$

No instantaneous cyclic dependency
“Not too hard” feedback

Output of FuelCmdOpen does not depend on $u_6$:

$$\text{FuelCmdOpen}(u_1, u_2, \ldots, u_7) = \frac{1}{14.7} (-0.366 + 0.08979 u_7 u_3 - 0.0337 u_7 u_3^2 + 0.0001 u_7^2 u_3)$$

No instantaneous cyclic dependency

Handled in recent work:
“Not too hard” feedback: solution

If we know the block’s **internals and input-output dependencies**:

![Diagram showing serial composition](image)

then this feedback reduces to serial composition:

![Diagram showing serial composition](image)
“Not too hard” feedback: solution

If we know the block’s **internals and input-output dependencies**:

![Diagram](image)

then this feedback reduces to serial composition:

![Diagram](image)

What if we don’t know the internals of the block?
How to define general feedback?

How to define feedback for any system $S$?
How to define general feedback?

How to define feedback for any system $S$?

A non-trivial problem.
How to define general feedback?

How to define feedback for any system $S$?

A non-trivial problem.

Even for deterministic and input-receptive systems.
Feedback for deterministic and input-receptive systems

How to distinguish invalid feedbacks:

from valid ones:

Solution [Malik ’94]: **fixpoint** semantics, starting with **unknown** values ("bottom" \( \perp \)).
Fixpoint semantics (also called *constructive* semantics)

\[ \neg \bot = \bot \]

Fixpoint stabilizes to \( \bot \): feedback is invalid.
Fixpoint semantics (also called *constructive* semantics)

\[ \neg \bot = \bot \]

Fixpoint stabilizes to \( \bot \): feedback is invalid.

\[ \bot \land 0 = 0 \]

Fixpoint stabilizes to \( 0 \): feedback is valid.
Limitations of constructive semantics

- It only applies to **deterministic and input-receptive** systems (i.e., total functions)
  
  *We also want to handle non-deterministic and non-input-receptive systems (partial relations).*
Limitations of constructive semantics

- It only applies to **deterministic and input-receptive** systems (i.e., total functions)
  
  *We also want to handle non-deterministic and non-input-receptive systems (partial relations).*

- There is **no refinement** in existing constructive semantics frameworks.
  
  *We not only have refinement, we also want refinement to be preserved by feedback.*
Preservation of refinement by composition

We want feedback to be **compositional**, i.e., for a composition operator $\circ$: 

*If $A \sqsubseteq A'$ (A' refines A) and $B \sqsubseteq B'$ then $A \circ B \sqsubseteq A' \circ B'$.*

This property is necessary for substitutability.
Preservation of refinement by composition

We want feedback to be *compositional*, i.e., for a composition operator $\circ$:

\[
\text{If } A \sqsubseteq A' \text{ (} A' \text{ refines } A \text{) and } B \sqsubseteq B' \text{ then } A \circ B \sqsubseteq A' \circ B'.
\]

This property is necessary for substitutability.

In particular, for feedback, we want:

\[
\text{If } A \sqsubseteq A' \text{ then } \text{feedback}(A) \sqsubseteq \text{feedback}(A').
\]
Preservation of refinement by composition

We want feedback to be *compositional*, i.e., for a composition operator $\circ$:

\[
\text{If } A \sqsubseteq A' \ (A' \text{ refines } A) \text{ and } B \sqsubseteq B' \text{ then } A \circ B \sqsubseteq A' \circ B'.
\]

This property is necessary for substitutability.

In particular, for feedback, we want:

\[
\text{If } A \sqsubseteq A' \text{ then feedback}(A) \sqsubseteq \text{feedback}(A').
\]

This is not easy to achieve – usual definitions don’t work [TOPLAS 2011, FPS 2014, LICS 2016].
Results of LICS 2016 paper

1. Two feedback operators:
   - **Instantaneous feedback**: applies to stateless (“memoryless”) systems.
   - **Feedback with unit-delay**: applies to stateful systems; instantaneous dependencies “broken” by a unit delay.

2. Both operators can handle non-deterministic and non-input-receptive systems.

3. Both operators proven to be compositional: they preserve refinement.
Two feedback operators:

- **Instantaneous feedback**: applies to stateless ("memoryless") systems.
- **Feedback with unit-delay**: applies to stateful systems; instantaneous dependencies “broken” by a unit delay.

Both operators can handle non-deterministic and non-input-receptive systems.

Both operators proven to be compositional: they preserve refinement.

In the special case of deterministic and input-receptive systems, both operators specialize to the standard solutions (instantaneous feedback specializes to constructive semantics).
Two feedback operators:

1. **Instantaneous feedback**: applies to stateless ("memoryless") systems.
2. **Feedback with unit-delay**: applies to stateful systems; instantaneous dependencies "broken" by a unit delay.

Both operators can handle non-deterministic and non-input-receptive systems.

Both operators proven to be compositional: they preserve refinement.

In the special case of deterministic and input-receptive systems, both operators specialize to the standard solutions (instantaneous feedback specializes to constructive semantics).

Serial composition = parallel composition followed by feedback.
This is a model of a hybrid automaton with polynomial dynamics, and an implementation of the 3rd model that appears in "Powertrain Control Verification Benchmark", 2014 Hybrid Systems: Computation and Control, X. Jin, J. V. Deshmukh, J. Kapinski, K. Ueda, and K. Butts.

Fuel Control System Model

This model uses only the ODEs to implement the dynamics.

Starup Mode

Power Mode Guard

\begin{align*}
&\theta [0 \ 90] \\
&\pi/30 (\text{rpm}) \to (\text{rad/s}) \\
&\text{engine speed (rpm)} \quad \boxed{[900,1100]} \\
&\text{throttle input (deg)} \quad \boxed{[0, 81.2]} \\
&1.1s+1 \quad \text{Throttle delay 1} \\
&8.8 \quad \text{Base opening angle} \\
&14.7 \quad \text{airbyfuel_ref} \\
&12.5 \\
\end{align*}

- **Simulink diagram**
  - **options (-fp, -ic, ...)**
  - **simulink2isabelle**
  - **Isabelle theory**
  - **RCRS theory**
  - **simplified MPT**
  - **compatibility check**
  - **Python simulation code**

*Another non-trivial problem:* Translation of arbitrary block diagrams to terms using the three primitive composition operators (serial, parallel, feedback).

Another non-trivial problem: *Translation of arbitrary block diagrams to terms using the three primitive composition operators (serial, parallel, feedback).*

Thank you – questions?

Bibliography:

A Theory of Synchronous Relational Interfaces.
ACM Transactions on Programming Languages and Systems (TOPLAS), 33(4), July 2011.

V. Preoteasa and S. Tripakis.
Refinement Calculus of Reactive Systems.

I. Dragomir, V. Preoteasa, and S. Tripakis.
Compositional Semantics and Analysis of Hierarchical Block Diagrams.

V. Preoteasa and S. Tripakis.
Towards Compositional Feedback in Non-Deterministic and Non-Input-Receptive Systems.
In 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), 2016.