Compositionality in the Science of System Design

Stavros Tripakis
Aalto University and UC Berkeley

Dagstuhl, Dec 5-6, 2016

The computer-controlled society
Example of a “smart” system (or CPS): autonomous intersection

Courtesy AIM project, CS Dept., UT Austin

Autonomous intersection: “real-life” version

Courtesy http://www.fastcodesign.com
Thanks to Christos Cassandras for recommending this video
How do we design such systems?

What do we even mean by “system”? 

System = state + dynamics (+ I/O)

A system: 

Another system: 

Another system: 

Another system: 

System = state + dynamics (+ I/O)

- Encompasses discrete systems, continuous systems, probabilistic systems, hybrid systems, finite-state systems, infinite-state systems, ...
- Not worrying about syntax at this point: automata, differential equations, enumerative, symbolic, ...
- Dynamical systems, but static are just a special case

- Mathematical model: transition system

Systems theory?

classic
Systems theory?

and many, many others ...

Complex systems?
Complex systems

Software!

```c
int x := get_input();
while x > 1 {
    if x is even
        x := x / 2;
    else
        x := 3*x + 1;
}
```

Collatz conjecture: program always terminates (open problem in mathematics)

How to design safety-critical systems?

- Trial and error
  - Un-scalable
  - Un-economic
  - Un-safe
  - Yet common...

Are the drivers supposed to debug the autopilot?

"It was described as a beta release. The system will learn over time and get better and that's exactly what it's doing. It will start to feel quite refined within a couple of months." – Elon Musk, Tesla CEO

Tesla autopilot video (source: youtube)
A better approach: Model-Based Design

Simulink, UML, SysML, HDLs, SystemC, ...

Describe the system that we want

Modeling

Analysis

Synthesis

Implement the system Automatically Correct-by-construction

Be sure that this is what we want
Semantics-preserving implementation of synchronous models (2003 – 2008)

Compositionality

(2008 – present)
Systems: **monolithic** definition

- System = state + dynamics (+ I/O)
- Dynamics = rules defining how state evolves in time

- Not very useful for complex systems ...

Systems: **compositional** definition

- System = collection of interacting subsystems
  - System = atomic system | composite system
  - Atomic system = state + dynamics (+ I/O)
  - Dynamics = rules defining evolution ...
  - Composite system = set of subsystems + composition
  - Composition = rules defining how subsystems interact

  system = state + dynamics + I/O + composition

Composition: fundamental in modern system theory!
Our work on compositionality (2008 – present)

- Modular code generation for synchronous languages [DATE’08, RTAS’08, POPL’09]
- Modular code generation for dataflow [ACM TECS’13]
- Co-simulation [EMSOFT’13, HSCC’15, SAMOS’15, Modelica’15, SAC’16, MEMOCODE’16]
- Multi-view modeling [TACAS’14, SAMOS’16, FACS’16]
- Compositional runtime enforcement [NFM’16]
- Verification and synthesis with component libraries [DATE’14, FACS’16]
- Compositional theories with refinement [EMSOFT’09, ACM TOPLAS’11, EMSOFT’14, SPIN’16, LICS’16]

See “Compositionality in the science of system design”, Proc. IEEE, 2016, for a survey.

The Refinement Calculus of Reactive Systems (RCRS)
http://rcrs.cs.aalto.fi/

Iulia Dragomir, Viorel Preoteasa, Stavros Tripakis
Aalto University and UC Berkeley
Motivation

• Compositional reasoning for reactive / embedded systems (e.g., Simulink)

RCRS = theory + toolset

Downloadable from http://rcrs.cs.aalto.fi/
RCRS theory

- **Relational interfaces** [EMSOFT'09, ACM TOPLAS'11]
  - Symbolic, synchronous version of interface automata [Alfaro, Henzinger]
  - Open, non-deterministic, non-input-complete systems
  - (this is crucial for static analysis)
  - Semantic foundation: relations
  - Limited to safety properties

- **Refinement calculus of reactive systems** [EMSOFT'14]
  - Richer semantics: predicate and property transformers
  - Can handle both safety and liveness properties

Types in Simulink

- No division by 0
- No sqrt of <0
- Double types
Relational interfaces

\[ u > \sqrt{u} \rightarrow x \]

\[ \text{Sqrt} \]

\[ \text{double} \rightarrow \text{double} \]

standard type

\[ u \geq 0 \land x = \sqrt{u} \]

relational interface

Can describe systems which are both non-deterministic and non-input-complete

Example of static analysis: catching incompatibility using symbolic methods

 чау

\[ u = -1 \]

\[ u \geq 0 \land \ldots \]

caught by taking the conjunction of the two formulas and checking satisfiability
A more tricky example

still incompatible, but not just conjunction of formulas

Serial composition involves $\forall – \exists$ quantification

$u \geq 0 \land \cdots$

$\forall u: (true \Rightarrow \exists x: u \geq 0 \land x = \sqrt{u})$
$\iff (true \Rightarrow u \geq 0)$
$\iff false$
Another example of static analysis: inferring new constraints on inputs (interface synthesis / type inference)

\[ u \geq 0 \]

RCRS toolset
Simulink is hierarchical

Hierarchical translation

- Atomic (library) blocks mapped to input/output formulas

\[ u > \sqrt{u} \rightarrow x \quad \Rightarrow \quad u \geq 0 \land x = \sqrt{u} \]

Tripakis
How to translate Simulink diagrams?

The algebra of block diagrams

- Only 3 primitive composition operators:
  - Serial
  - Parallel
  - Feedback

Defining feedback: non-trivial [LICS 2016]
Translation is also non-trivial [SPIN 2016]

Our Translator implements 3 different translation strategies

Interesting questions

• Are the different algebra terms semantically equivalent?
  – For instance:

  \[ A \circ B = feedback(A || B) ? \]
  – This is now proven formally in Isabelle [Arxiv’16]

• Which terms are “better”?
  – [SPIN’16] provides experimental comparisons
After translation: expansion + simplification

- **Expansion**: replace all atomic components and composition operators with their definitions
  - This usually results in a **BIG** formula

- **Simplification**: simplify the formula
  - This generally involves quantifier elimination
  - Luckily it is relatively easy to do for Simulink because all blocks are deterministic (i.e., partial functions)
  - And also because of hierarchy!

- 2000+ lines of Isabelle code
Example of expansion/simplification

\[ u = v + 1 \quad \text{if} \quad u \geq 0 \]

\[ v \geq -1 \]

 Expansion/simplification: another example

**Translation:**

feedback\(((\text{Add} \parallel \text{Id}) \circ \text{UnitDelay} \circ (\text{Split} \parallel \text{Id}))\)

**Expansion and simplification:**

\[ (e, s) \rightsquigarrow (a := s, s' := s + e) \]
Does it work on real systems?

- Yes!
- Several real-life benchmarks
- Publicly available Simulink models provided by Toyota ([http://cps-vo.org/group/ARCH/benchmarks](http://cps-vo.org/group/ARCH/benchmarks))
- Automotive fuel control system:
  - 3-level hierarchy, 104 blocks, 7 subsystems, non-linear arithmetic, control, bool/int/real vars, ...
  - Translation time: negligible (any strategy)
  - Expansion/simplification time: 1 min to 1 hour, depending on which term was used
  - Top-level simplified formula: many pages long; but no internal variables!
Open (?) problems 1

• Looking for effective symbolic procedures for:
  – Formula simplification
  – Quantifier elimination
  – SAT solving
  – ...

Open (?) problems 2

• In the end, we get a transition relation parameterized by $\Delta t$ (plus a precondition, but let’s ignore this right now):
  – $\phi(x, y, s, s', \Delta t)$
• Suppose there are no inputs (i.e., no $x$) and let $S_0$ be a set of initial states.
• Let $Behaviors(\phi, S_0)$ be the set of trajectories starting from $S_0$. This is also parameterized by $\Delta t$.
• Questions:
  – Can we prove, and how, that $Behaviors(\phi, S_0)$ satisfies a given safety/liveness property, for all $\Delta t > 0$?
  – Would this imply that the continuous system satisfies the same property?
Refinement

Incremental design

Suppose we have designed and verified this “steer-by-wire” system:

\[ v \in [v_{\text{min}}, v_{\text{max}}] \]

\[ \text{latency} \leq 10\text{ms} \]
Incremental design

Suppose we want to replace B with Z:

How to ensure properties are preserved (substitutability)?
Refinement theories

Z \leq B: Z refines B

(1) If A' \leq A and A satisfies P then A' satisfies P.
(2) If A' \leq A and B' \leq B, then A' \cdot B' \leq A \cdot B.

Z \leq B and (1) and (2) => substitutability!

Refinement relation

\phi' \leq \phi \overset{\text{def}}{=} \begin{cases} \text{in} (\phi) \Rightarrow \text{in} (\phi') \\ (\text{in} (\phi) \wedge \phi') \Rightarrow \phi \end{cases}

\text{in}(\phi) \overset{\text{def}}{=} \exists \text{outputs: } \phi

- Refinement <=> substitutability:
  - A' can replace A in any context iff A' \leq A.
  - i.e., refinement both necessary and sufficient condition for substitutability.
  - Note: sometimes the strongest contravariance condition \phi' \Rightarrow \phi is used, which is sufficient but not necessary.
Handling components with state

\[ u \xrightarrow{z'} x \]

Delay

\[ x = s \land s' = u \]

\( s \): state variable
\( s' \): next state variable

Expressing temporal properties in LTL (including liveness)

\[ A \quad \square (x \geq 0) \quad x \quad B \quad \square \diamond (x = 1) \]
Thank you

Questions?