Refinement calculus for reactive systems

Stavros Tripakis
UC Berkeley and Aalto University

Joint work with Viorel Preoteasa (Aalto), Ben Lickly (Berkeley), Thomas Henzinger (IST Austria), and Edward Lee (Berkeley)

Supported by NSF projects COSMOI and ExCAPE
Motivations

- Component-based design
- Incremental design and verification
- Behavioral type theories
- Automatic synthesis of abstractions
Incremental verification

A “steer-by-wire” system:

\[
\nu \in [\nu_{\text{min}}, \nu_{\text{max}}]
\]

\text{latency} \leq 10\text{ms}
Incremental verification

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

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

How to ensure properties are preserved?

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

$latency \leq 10ms$
Refinement theories
(e.g., interface theories [Alfaro, Henzinger et al.])

• **Interface** = component abstraction
• **Interface composition**: $A \bullet B = C$
• **Interface refinement**: $A' \leq A$

• Theorems:

(1) If $A' \leq A$ and $A$ satisfies $P$ then $A'$ satisfies $P$.
(2) If $A' \leq A$ and $B' \leq B$, then $A' \bullet B' \leq A \bullet B$. 
Incremental verification with refinement theories

(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) $\Rightarrow$ substitutability!
Refinement theories
(e.g., interface theories [Alfaro, Henzinger et al.])

- **Interface** = component abstraction
- **Interface composition**: $A \bullet B = C$
- **Interface refinement**: $A' \leq A$

**Theorems:**

1. If $A' \leq A$ and $A$ satisfies $P$ then $A'$ satisfies $P$.
2. If $A' \leq A$ and $B' \leq B$, then $A' \bullet B' \leq A \bullet B$.

Note: composition is **partial** => can specify **compatibility** (local property)
Motivation #2: refinement theories = behavioral type theories

• Type checking:
  – A very successful, “light-weight” analysis
  – No specification required
    • Contrast to verification
  – Standard practice in software
  – But relatively limited types
Type checking in Simulink

no division by 0?

no sqrt of <0?

double

double
Earlier work: relational interfaces (behavioral, richer types)

\[ u \geq 0 \land x^2 = u \]

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

standard type

relational interface (behavioral type)
Earlier work: relational interfaces
(behavioral, richer types)

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

Note: this is conjunction, not implication

[ACM TOPLAS 2011]
Catching incompatibility

caught by taking simply the conjunction of the two formulas

\[ u = -1 \quad \text{and} \quad u \geq 0 \land \cdots \]
What about this example?

This is not just conjunction of formulas.
Catching incompatibility

∀u: (true ⇒ ∃x: u ≥ 0 ∧ x^2 = u) ≡ false

game
In general: “demonic” serial composition

Standard: \( \phi := \phi_1 \land \phi_2 \)

“Demonic”: \( \phi := \phi_1 \land \phi_2 \land (\forall y : \phi_1 \Rightarrow in(\phi_2)) \)

\( in(\phi_2) := \exists z : \phi_2 \)

Tripakis

Computing \( \forall \) can often be avoided or delayed [SPIN 2013]
Bottom-up interface synthesis = automatic abstraction

Given interfaces for A, B, C, synthesize automatically new interface for P (and also check compatibility in the process)
Inferring new constraints on inputs

\[ v \geq -1 \]

\[ u = v + 1 \]

\[ u \geq 0 \]

Tripakis
Handling components with state

$s : \text{state variable}$

$u \xrightarrow{z^{-1}} x$

$$x = s \land s' = u$$
Finite-state relational interface

Static interface:
(holds at every round)
\[
\neg (\text{empty} \land \text{full}) \\
\land \\
\neg (\text{write} \land \text{read}) \\
\land \\
\text{empty} \Rightarrow \neg \text{read} \\
\land \\
\text{full} \Rightarrow \neg \text{write}
\]

Dynamic (state-dependent) interface:
Recent work: refinement calculus for reactive systems [EMSOFT 2014]

- Relational interfaces limited to safety properties
- What about liveness?
- Answer: refinement calculus for reactive systems
Refinement calculus for reactive systems

• Example:

Are blocks A and B compatible?

Yes! New constraint inferred: □ ◇ x
Refinement calculus for reactive systems

• Inspired from Refinement Calculus:
  – Well-established theory for sequential programs
    [Dijkstra, Ralph J. Back, ...]
  – Semantics: weakest preconditions
  – Programs = predicate transformers
    • Given set of post-states, return set of pre-states

• Reactive systems = property transformers:
  given set of infinite sequences of outputs,
  return set of infinite sequences of inputs

Implementation in Isabelle publicly available
Refinement (behavioral subtyping)

(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) $\implies$ substitutability
Refinement

\[ \phi' \leq \phi \, \overset{\text{def}}{=} \ (\text{in}(\phi) \Rightarrow \text{in}(\phi')) \]

\[ (\text{in}(\phi) \land \phi') \Rightarrow \phi \]

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

• Refinement \( \leq \) substitutability:

\[ A' \text{ can replace } A \text{ in any context iff } A' \leq A. \]

• i.e., refinement both necessary and sufficient condition for substitutability.

• Previous similar notions (e.g., \textit{Liskov-Wing behavioral subtyping}) are sufficient but not necessary.
Conclusions

• Refinement calculus for reactive systems: generic framework for compositional reasoning

• Components described by formulas (I/O predicates, LTL, symbolic transition systems, ...)

• Illegal inputs => Compatibility checking

• Refinement = substitutability

• Implementation
  – Theory available in Isabelle theorem prover
  – Simulink front-end under implementation