Relational Interfaces and Refinement Calculus for Compositional System Reasoning

Viorel Preoteasa

Joint work with Stavros Tripakis and Iulia Dragomir

Overview

- Motivation
- General refinement
- Relational interfaces
- Refinement calculus for reactive systems
- Liveness properties
- Modeling Simulink Diagrams

Motivation

- Is the system correct?
- Can we replace a subsystem by another subsystem, preserving the functionality?
- Compatibility. Is the composition of two systems meaningful?
- Can we model liveness properties?

We are interested in reactive systems – systems that repeatedly take some input from the environment and produce some output

Refinement

Refinement (denoted $A \sqsubseteq B$):

- System A is refined by system B or
- Informally: B can replace A in any context
- Formally:
  1. If A satisfies a property P then B satisfies P
  2. If $A \sqsubseteq A'$ and $B \sqsubseteq B'$ then $A \bullet B \sqsubseteq A' \bullet B'$

  $A \bullet B$ denotes some composition of systems A and B

Refinement

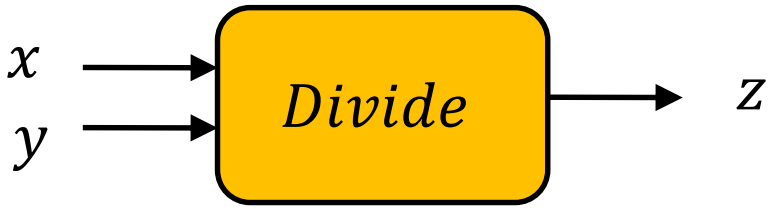
- Correctness:
  - Specification $\sqsubseteq$ Implementation
- Substitutability:
  - If we have $B \sqsubseteq B'$, then
  - $A \bullet B \bullet C \sqsubseteq A \bullet B' \bullet C$
  - The system $A \bullet B' \bullet C$ satisfies all properties satisfied by $A \bullet B \bullet C$
- (In)Compatibility:
  - $A \bullet B = Fail$ or $A \bullet B \sqsubseteq Fail$ where
  - $Fail$ = while true do skip, or $Fail$ = unhandled exception, or $Fail$ = assertion on input is false for every input

Interface theories

- Interface theories can express some of the properties presented above, but not liveness
  - Relational interface introduced by Tripakis et al, *A Theory of Synchronous Relational Interfaces,* ACM TOPLAS, 2011
  - Interface automata introduced by Alfaro et al, *Interface Automata*, FSE, ACM, 2009
- On the other hand there are frameworks capable of expressing liveness properties, but they cannot express compatibility of systems.
  - Focus framework, Broy et al, *Specification and development of interactive systems: focus on streams interfaces and refienemt*, Springer, 2001

Relational Interfaces - Example

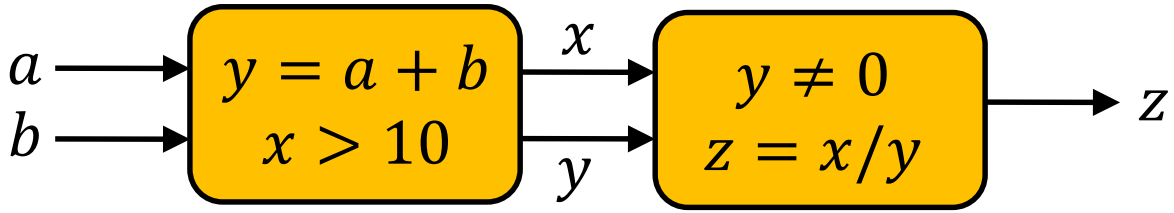
- Division component:



- Contract: $y \neq 0 \wedge z = x/y$
- The condition $y \neq 0$ introduces a requirement on input y
- If input $y = 0$, then $Divide$ fails (this is different from $Fail$ = fails for all inputs).

Relational Interfaces – Composition

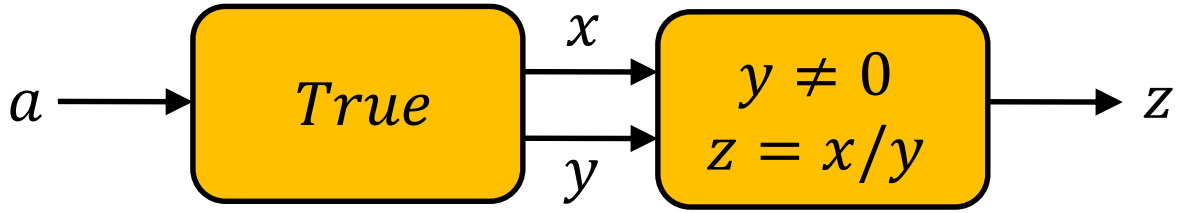
- Output of one component becomes the input of the second component



- The requirement on y is propagated to a and b
- Choosing a and b properly we can ensure $y \neq 0$
- The composition fails if $a = -b$ (the composition is not $Fail$)

Relational Interfaces - Incompatibility

- The two systems are incompatible



- The component $True$ produces non-deterministically values x and y
- By controlling a there is no possibility of ensuring $y \neq 0$
- The composition of these systems is $Fail$, because the composition fails for every input.

Relational Interfaces – Limitations

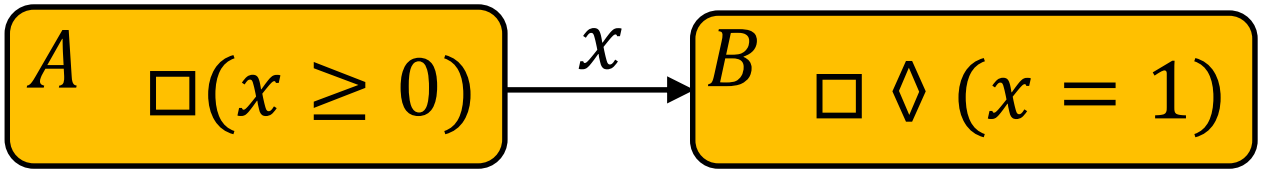
- Relational interfaces cannot model liveness properties
- Semantics of relational interfaces:
  - prefix closed sets of finite input output traces

Reactive systems

- A reactive system is a machine that takes as input an infinite sequence $x_0, x_1, x_2, \ldots$ and it outputs an infinite sequence $y_0, y_1, y_2, \ldots$
- Assume a system that counts and outputs how many input values seen so far are true.
- Then
  - Input: $0,1,0,0,1,1,1,0,0, \ldots$
  - Output: $0,1,1,1,2,3,4,4,4, \ldots$

Our Goal

A compositional theory for reactive systems with both safety and liveness



- A specifies that its output x is always greater or equal than zero
- B requires that its input is infinitely often equal to one.
- The output of A is connected to the input of B.
- In our framework: these components are incompatible
- We want to be able to use LTL formulas in specifications

Refinement Calculus for Reactive Systems

- **Monotonic property transformers**
  - Functions mapping sets of infinite output sequences into sets of output sequences
  - Property = set of infinite sequences
- A system A applied to a set of output sequences Q is the set of all input sequences that do not fail and produce an output sequence in Q.
- Based on Refinement Calculus introduced by Back, *On the correctness of refinement in program development*, 1978

Refinement Calculus for Reactive Systems

This semantics enables reasoning about all features that we mentioned at the beginning:

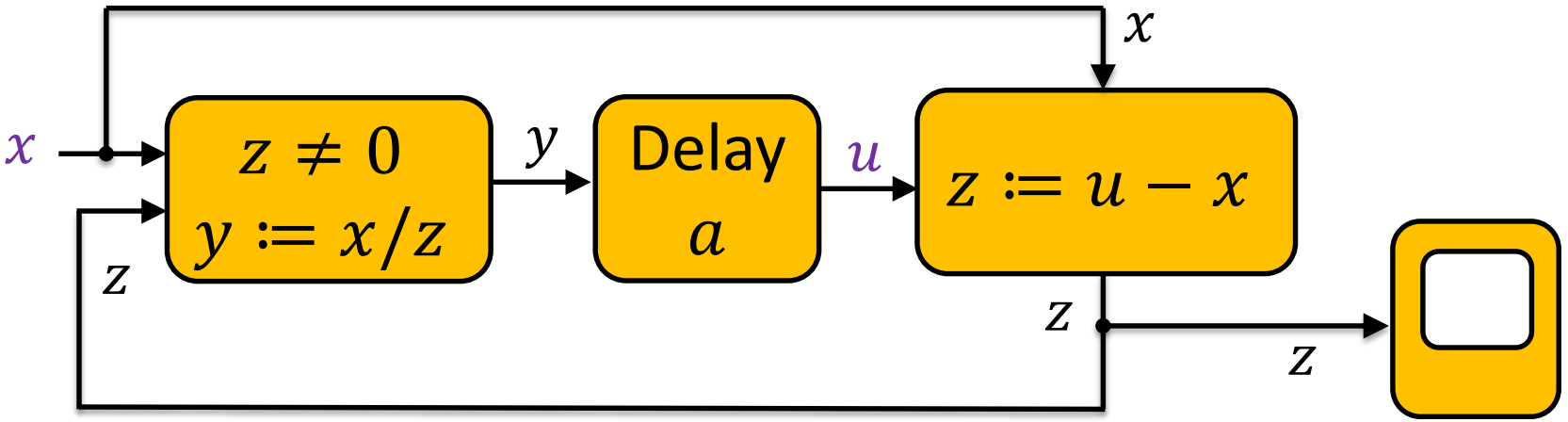
- Correctness
- Substitutability
- Compatibility
- And also liveness properties

Reactive systems – Operations

The operations on reactive systems are defined in the same way as for predicate transformers

- Sequential composition = function composition:
  - $(A \circ B)(Q) = A\big(B(Q)\big)$
  - where Q is a set of infinite sequences.
- Refinement = point-wise subset:
  - $A \sqsubseteq B \Leftrightarrow (\forall Q : A(Q) \subseteq B(Q))$
- $Fail(Q) = \emptyset$

Simulink Example



- $t = 0$: x_0; $u_0 := a$; $z_0 := u_0 - x_0$; $y_0 := x_0/z_0$; $z_0 = u_0 - x_0 \neq 0$
- $t = 1$: x_1; $u_1 := y_0$; $z_1 := u_1 - x_1$; $y_1 := x_1/z_1$; $z_1 = u_1 - x_1 \neq 0$

Simulink Example

- The variable u after the delay is calculated by:

$$u_0 := a; \quad u_{n+1} := x_n/(un - xn)$$

- The output is given by:

$$z_n := un - xn$$

- The input x_n must satisfy the following property:

$$(\forall n : u_n \neq xn)$$

Simulink Example as Property Transformer

- Our tool produces the following property transformer

$$\{\forall u: (u_0 = a) \wedge \left(\forall n : u_{n+1} = \frac{x_n}{u_n - xn}\right) \Rightarrow (\forall n : un \neq xn)\}$$

  - $[z : \exists u : u = a \wedge \Box\, (u^1 = \frac{x}{u-x} \wedge z = u - x)]$

Simulink Example as Property Transformer

- Using Linear Temporal Logic

$$\{\forall u : u = a \wedge \square\left(u^1 = \frac{x}{u-x}\right) \Rightarrow \square(u \neq x)\} \circ$$

$$[z : \exists u : u = a \wedge \square\,(u^1 = \frac{x}{u-x} \wedge z = u - x)]$$

Conclusions

- We can model a number of desired features
  - Correctness
  - Substitutability
  - Compatibility
  - Liveness properties
  - … and many more
- We can use linear temporal logic to specify and reason about these systems
- We built a tool that translates Simulink models to property transformers.
- The results were formalized in Isabelle theorem prover