Interface-based Design of Embedded Systems

Thomas A. Henzinger
University of California, Berkeley
Interface-based Design
Interface-based Design
Interface-based Design
Interface-based Design
Compositional Component Models

If $A\parallel B$ is defined and $A \leq a$ and $B \leq b$, then $a\parallel b$ is defined and $A\parallel B \leq a\parallel b$.

enable independent component verification

Compositional Interface Models

If $a\parallel b$ is defined and $A \leq a$ and $B \leq b$, then $A\parallel B$ is defined and $A\parallel B \leq a\parallel b$.

enable independent interface implementation
A Component Model

\[ x \in \text{Nat} \land y \in \text{Nat}\{0\} \Rightarrow z = x \div y \]

-(mis)behaves in every environment
-examples: circuit; executable code

An Interface Model

\[ x \in \text{Nat} \land y \in \text{Nat}\{0\} \land z \in \text{Nat} \]

-constrains the environment
-example: type declaration
The Component Model

∀x,y. ∃z. ( x ∈ Nat ∧ y ∈ Nat\{0} ⇒ z = x÷y )

input-universal (adversarial environment)

The Interface Model

∃x,y. ∃z. ( x ∈ Nat ∧ y ∈ Nat\{0} ∧ z ∈ Nat )

input-existential (helpful environment)
The Interface Model

Input assumption

Prescriptive:

“How can the component be put together with other components?”
Propagation of Environment Constraints

\[ x=0 \implies y=0 \]

true
Propagation of Environment Constraints

$x=0 \Rightarrow y=0 \quad \text{true}$
Propagation of Environment Constraints

\[ x=0 \Rightarrow y=0 \quad \text{true} \]

\[ y = 0 \]

\[ \forall x, z. ( \text{true} \land x = z \Rightarrow ( x=0 \Rightarrow y=0 )) \]
Propagation of Environment Constraints

\[ y = 0 \quad \text{true} \]

The resulting interface.
Propagation of Environment Constraints

\[ y = 0 \quad \text{true} \]

Illegal connection.
Stateless interface models (traditional “types”): value constraints

Stateful interface models (“behavioral types”): temporal ordering constraints, real-time constraints, etc.
A Component Model: I/O Automata

This is an illegal component, because it is not prepared to accept input b.

[Lynch, also Lamport, Alur/H]
Another Component Model: CSP

Composition may lead to deadlocks, and requires verification if this is undesirable.

[Hoare, also Milner, Harel]
An Interface Model: Interface Automata

These interfaces are incompatible, because the receiver expects the environment to provide input b.

[de Alfaro/H, also Dill]
Component Models

- composition || is conjunction/product
- abstraction ≤ is covariant

Interface Models

- composition || is game-theoretic
- implementation ≤ is contravariant
Incompatible product state, but environment can prevent this state.
The Composite Interface.
The Composite Interface.
Computing the Composite Interface

- Construct product automaton.
- Mark deadlock states as incompatible.
- Until no more incompatible states can be added: mark state \( q \) as incompatible if the environment cannot prevent an incompatible state to be entered from \( q \).
- If the initial state is incompatible, then the two interfaces are incompatible. Otherwise, the composite interface is the product automaton without the incompatible states.

This computes the states from which the environment has a strategy to avoid deadlock. The propagated environment constraint is that it will apply such a strategy.
Component Abstraction

Abstraction is implication (simulation; trace containment).
Interface Implementation

$x \in \text{Even}$        $x \in \text{Nat}$

$\downarrow$

VI

$x \in \text{Nat}$        $x \in \text{Odd}$

$\uparrow$

Implementation is I/O contravariant.
Interface Implementation

Implementation must obey output guarantee.

- $x \in \text{Nat}$
- $x \in \text{Odd}$
- $x \in \text{Nat}$
- $x \in \text{Nat}$
Interface Implementation

Implementation must accept all permissible inputs.
Alternating Simulation

$Q \leq q$

iff

1. for all inputs $i$, if $q \rightarrow_{i} q'$, then there exists $Q'$ such that $Q \rightarrow_{i} Q'$ and $Q' \leq q'$,

and

2. for all outputs $o$, if $Q \rightarrow_{o} Q'$, then there exists $q'$ such that $q \rightarrow_{o} q'$ and $Q' \leq q'$.

If there is a helpful environment at $q$, then there is a helpful environment at $Q$ [Alur/H/Kupferman/Vardi].
Algorithms & Tools

- **interface compatibility** (reachability game) can be checked in linear time

- **interface implementation** (alternating simulation) can be checked in quadratic time

We are currently implementing this in JBuilder [Chakrabarti/de Alfaro/H/Jurdzinski/Mang].