Relational Interfaces

Stavros Tripakis

UC Berkeley

Joint work with Ben Lickly,
Tom Henzinger and Edward Lee

Component-Based Design

- How can we build large, complex systems from smaller, simpler systems?
  - We call the latter **components**
- Raises many interesting questions:
  - What kind of components do we need?
    - What are the right building blocks?
  - Which components to use and how to connect them?
  - What is a component? How to reason about components?

Interface theories [e.g., Alfaro, Henzinger, et al.]

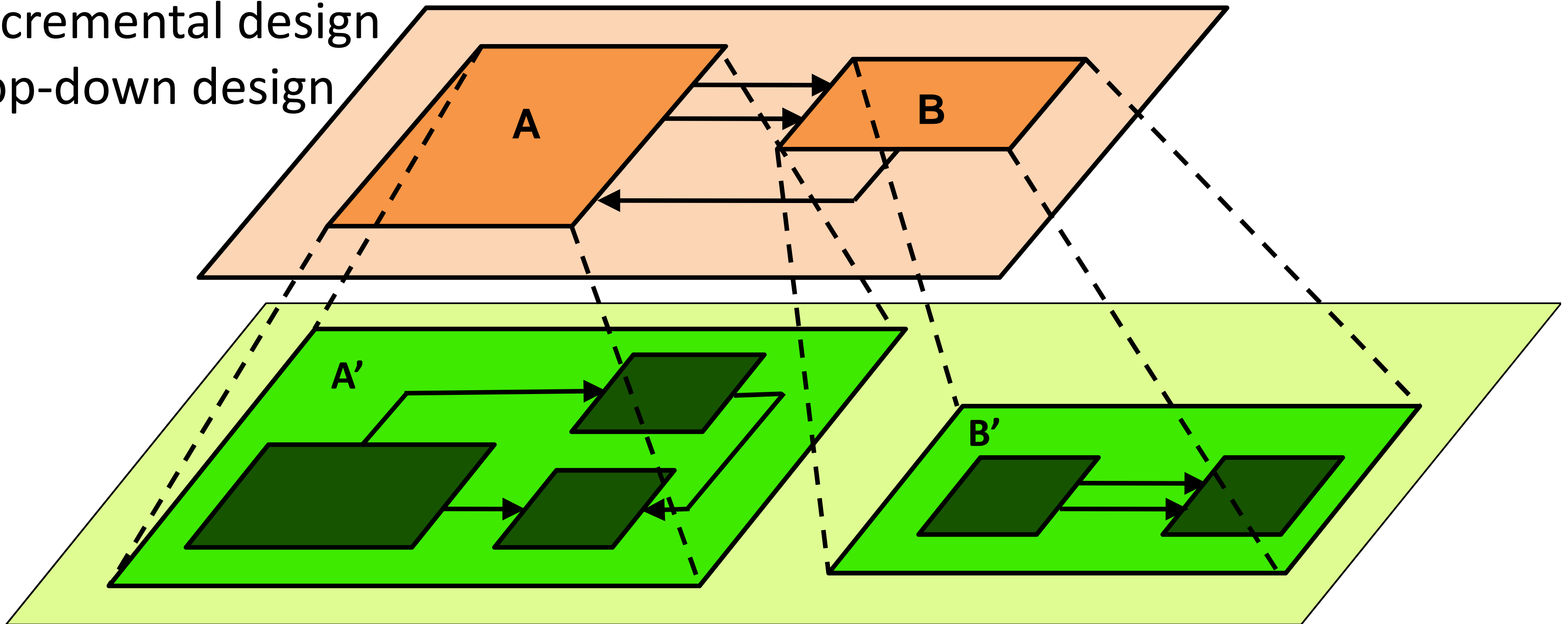
- Interface = component abstraction
- Interface composition: A • B = C
- Interface refinement: A’ $\leq$ A
- Theorems:

(1) If A’ $\leq$ A and A satisfies P then A’ satisfies P.

(2) If A’ $\leq$ A and B’ $\leq$ B, then A’ • B’ $\leq$ A • B.

Substitutability

- Incremental design
- Top-down design

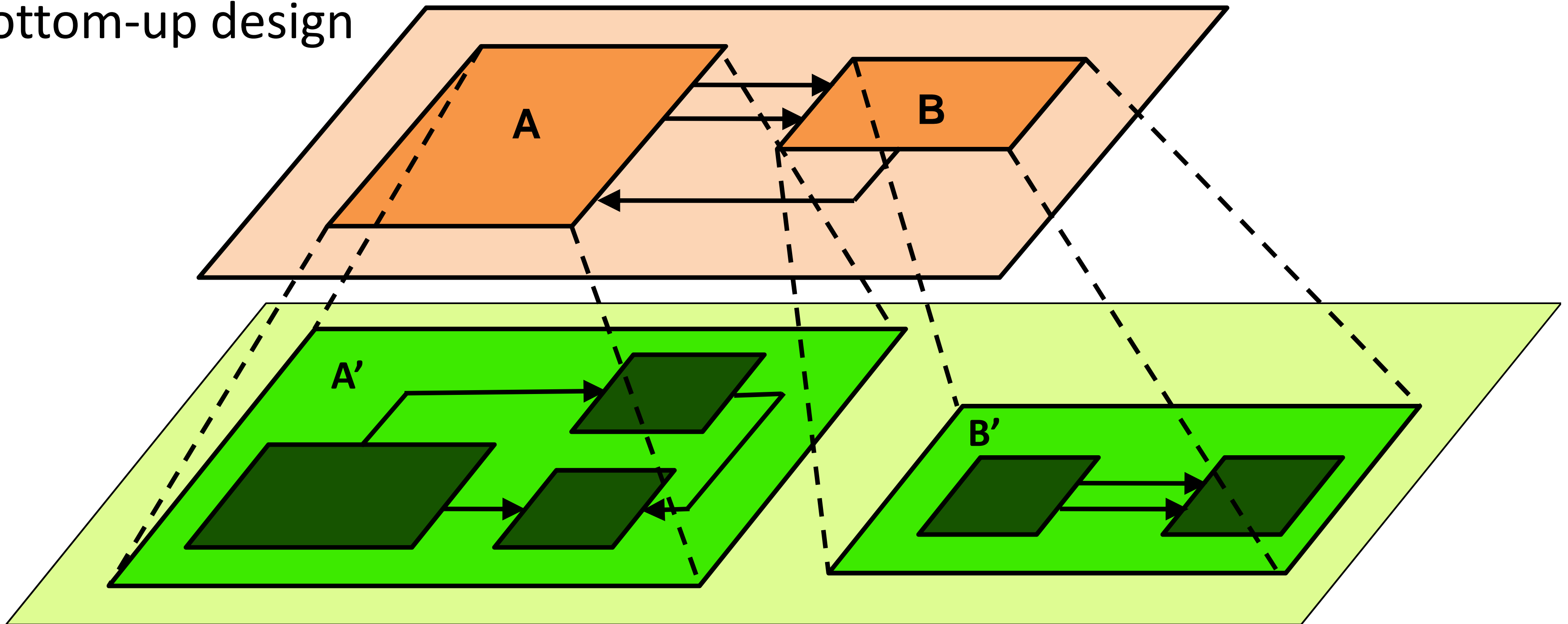


(1) If A’ ≤ A and A satisfies P then A’ satisfies P.

(2) If A’ ≤ A and B’ ≤ B, then A’ • B’ ≤ A • B.

Synthesis of abstractions

- Bottom-up design



If A and B are interfaces then we can compute an interface for their composition: A • B.

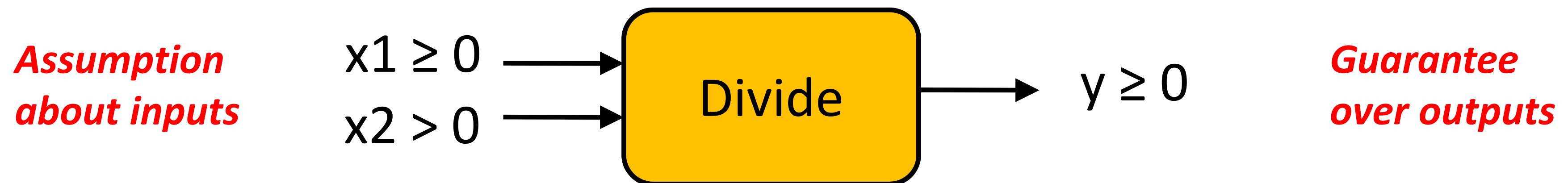
Tons of related work …

- Floyd, Hoare, Dijkstra, Wirth, …, 1960s, 1970s, …: pre/post-conditions, stepwise refinement, …
- Abrial, 1980s, 1990s: the Z notation, the B method
- Back, 1980s, …: refinement calculus
- Liskov, 1980s: Modular program construction using abstractions
- Meyer, 1980s: Eiffel, contracts (pre/post-conditions), subcontracting (inheritance)
- Lynch, 1980s: I/O automata
- Dill, 1980s: Trace theory for automatic hierarchical verification of speed-independent circuits
- Misra/Chandy, Jones, Barringer/Kuiper/Pnueli, Stark, …, many others, 1980s, 1990s, 2000s, …: compositional verification, assume-guarantee, …
- Broy, 1990s, …: FOCUS
- Software engineering: software reuse, modularization, Parnas, many others, …
- Type theory: covariance/contravariance
- …

Non-relational Interfaces

e.g., [Doyen et al., EMSOFT'08]

- **Separate** predicates over inputs and outputs



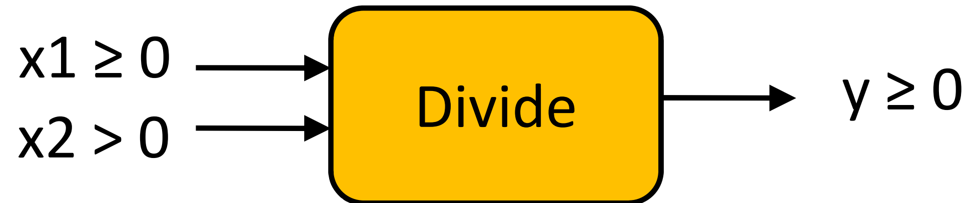
- Cannot express input-output **relations**:

$$y = x1/x2$$

Relational Interfaces

[this work]

- Predicates over **both** inputs and outputs



- Can express input-output **relations**:

$$x_2 \neq 0 \wedge y = \frac{x_1}{x_2}$$

deterministic (function)

$$x_2 \neq 0 \rightarrow y = \frac{x_1}{x_2}$$

non-deterministic (relation)

Plan of talk

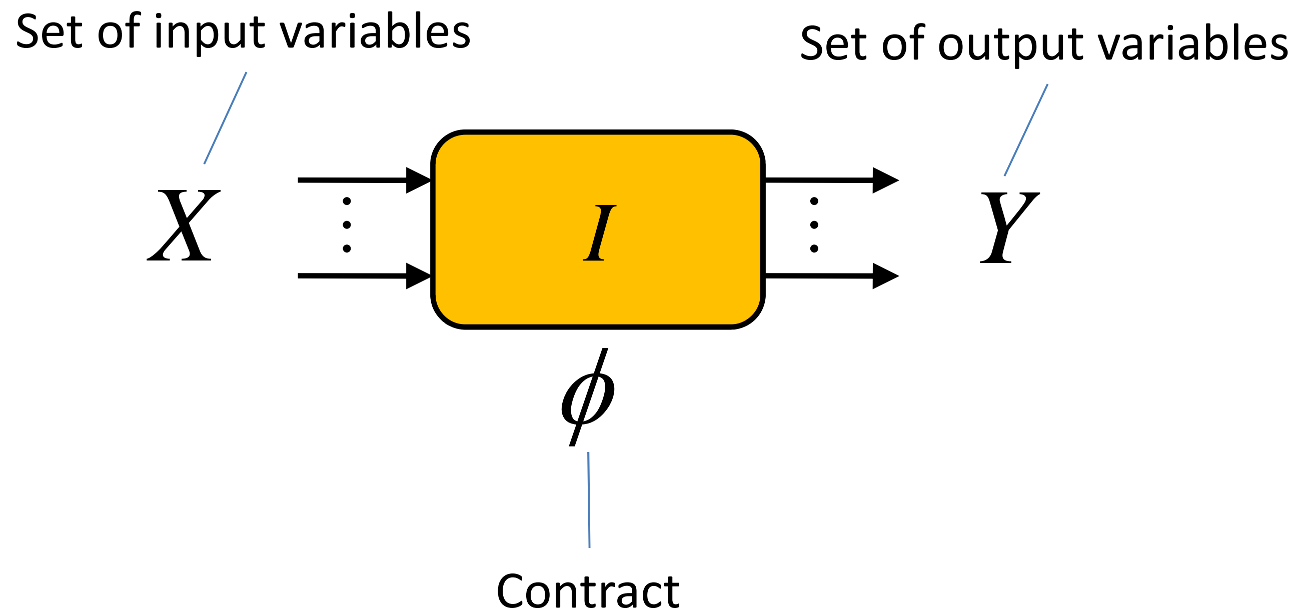
- Relational interfaces
  - Stateless, stateful
- Environments and pluggability
- Refinement
  - Refinement and pluggability
- Composition
  - Connection, feedback
  - Preservation of refinement by composition

Plan of talk

- Relational interfaces
  - Stateless, stateful
- Environments and pluggability
- Refinement
  - Refinement and pluggability
- Composition
  - Connection, feedback
  - Preservation of refinement by composition

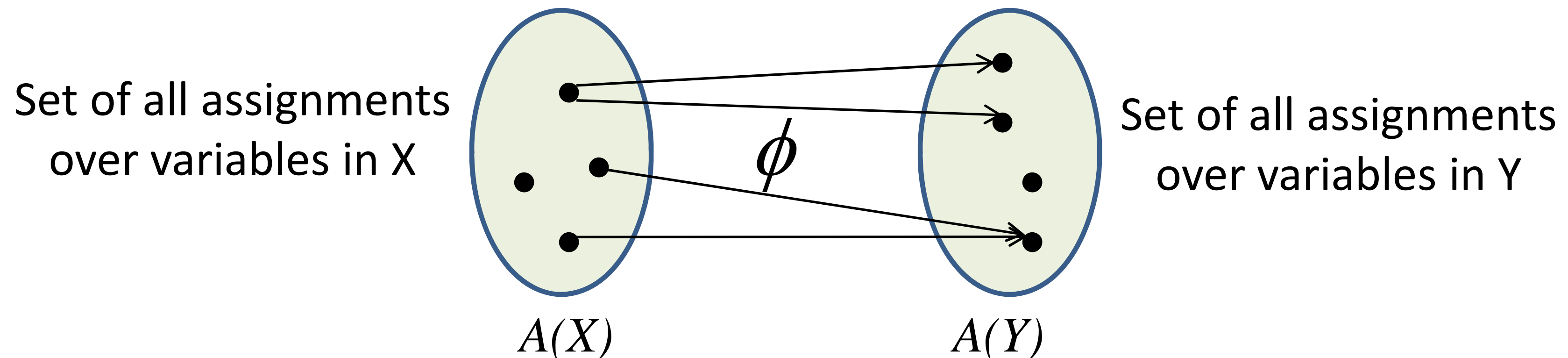
Stateless Relational Interfaces

$$I = (X, Y, \phi)$$



Contracts

- Semantically: **relations** between input and output assignments: $\phi \subseteq A(X) \times A(Y) = A(X \cup Y)$



- Syntactically: **predicates** or something similar

$$x_2 \neq 0 \wedge y = \frac{x_1}{x_2}$$

Assumptions and Guarantees

- Input **assumptions**: set of legal input assignments

$$\boxed{in(\phi) \equiv \exists Y : \phi} \subseteq A(X)$$

- Output **guarantees**: set of possible output assignments

$$\boxed{out(\phi) \equiv \exists X : \phi} \subseteq A(Y)$$

Assumptions and Guarantees

- Input **assumptions**: set of legal input assignments

$$in(\phi) \equiv \exists Y : \phi$$

$$in(x_2 \neq 0 \wedge y = \frac{x_1}{x_2}) \equiv x_2 \neq 0$$

$$in(x_2 \neq 0 \rightarrow y = \frac{x_1}{x_2}) \equiv true$$

Stateful Relational Interfaces

$$I = (X, Y, \xi)$$

$$\xi : A(X \cup Y)^* \rightarrow C(X \cup Y)$$

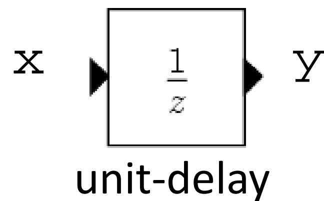
Set of all possible states over X U Y

state = history = *a1 a2 … ak*

Set of all possible contracts over X U Y

Stateless = special case of stateful
= same contract at all states

Example of stateful interface: unit delay




```
x:  0   1   2   3   4   5  ...
y:  v0  0   1   2   3   4  ...
```

$$I_{ud} = (\{x\}, \{y\}, \xi_{ud})$$

$$\xi_{ud}(\varepsilon) \equiv (y = v_0)$$
$$\xi_{ud}(s \cdot a) \equiv (y = a(x))$$

initial state

last step

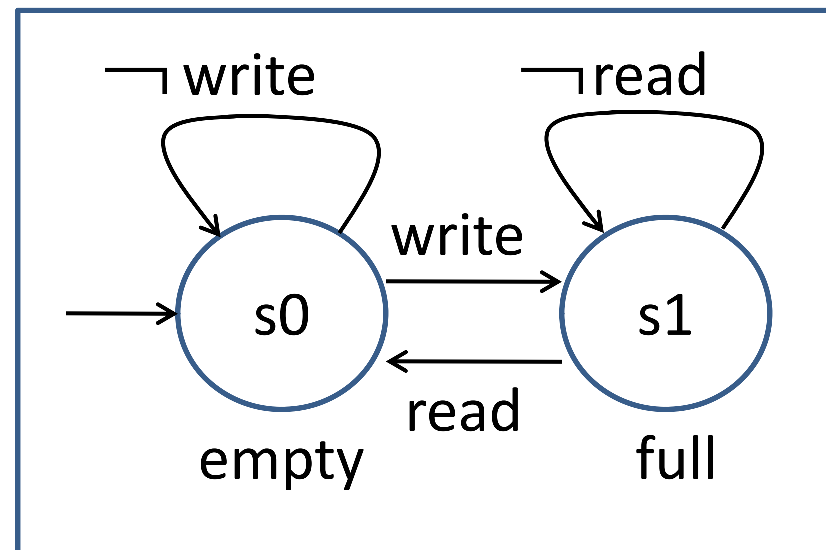
Infinite-state interface

Example of finite-state interface: 1-place buffer

Note: this says almost nothing about implementation

Note: this says nothing about data

$$\neg(\text{empty} \wedge \text{full})$$
$$\wedge$$
$$\neg(\text{write} \wedge \text{read})$$
$$\wedge$$
$$\text{empty} \rightarrow \neg\text{read}$$
$$\wedge$$
$$\text{full} \rightarrow \neg\text{write}$$



Well-formed and well-formable interfaces

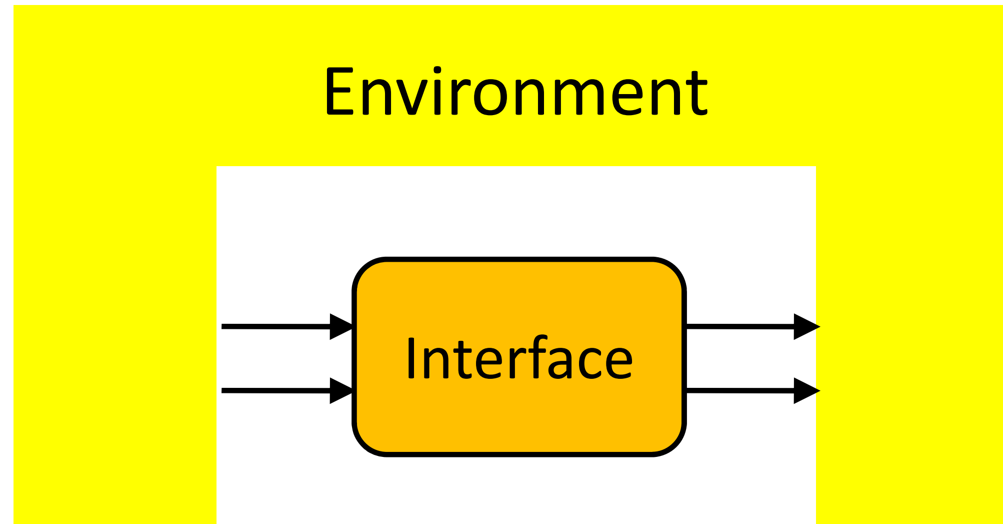
- Well-formed:
  - Every reachable state has a satisfiable contract
- Well-formable:
  - Can be made well-formed by restricting the inputs
  - Amounts to finding a winning strategy in a game [Alfaro-Henzinger '01, Dill '89, Back '90]
- For stateless interfaces,

  well-formed = well-formable = satisfiable

Plan of talk

- Relational interfaces
  - Stateless, stateful
- Environments and pluggability
- Refinement
  - Refinement and pluggability
- Composition
  - Connection, feedback
  - Preservation of refinement by composition

Environments and Pluggability

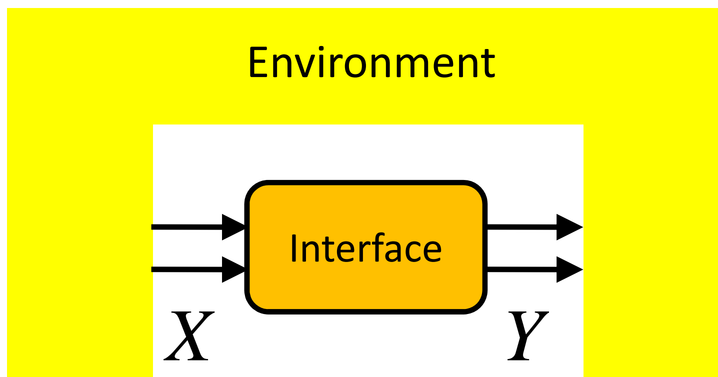


Environments

$$E = (X, Y, \phi_X, \phi_Y)$$

predicate on X
(possible inputs)

predicate on Y
(desirable outputs)



Think precondition/postcondition

Pluggability

$$I = (X, Y, \phi)$$

$$E = (X, Y, \phi_X, \phi_Y)$$

- Interface I is pluggable to environment E if:

$$\forall X : \phi_X \rightarrow in(\phi)$$

$$\forall X, Y : \phi_X \wedge \phi \rightarrow \phi_Y$$

Plan of talk

- Relational interfaces
  - Stateless, stateful
- Environments and pluggability
- Refinement
  - Refinement and pluggability
- Composition
  - Connection, feedback
  - Preservation of refinement by composition

Refinement

$$I' = (X, Y, \phi') \quad \leq \quad I = (X, Y, \phi)$$

iff

$$\forall X : in(\phi) \rightarrow in(\phi')$$

$$\forall X, Y : in(\phi) \wedge \phi' \rightarrow \phi$$

Refinement examples

$$\forall X : in(\phi) \rightarrow in(\phi')$$
$$\forall X, Y : in(\phi) \wedge \phi' \rightarrow \phi$$

more deterministic outputs

$$x = y \qquad \leq \qquad x = y \vee x + 1 = y$$

more legal inputs

$$x_2 \neq 0 \rightarrow y = \frac{x_1}{x_2} \qquad \leq \qquad x_2 \neq 0 \wedge y = \frac{x_1}{x_2}$$

or both

$$x = y \qquad \leq \qquad x > 0 \wedge (x = y \vee x + 1 = y)$$

Refinement properties

- Reflexive, transitive, antisymmetric: partial order
- Top element: *false*

$$false \rightarrow in(\phi')$$
$$false \wedge \phi' \rightarrow false$$

- No bottom element
  - *true* is not bottom:

$$in(\phi) \rightarrow true$$
$$in(\phi) \wedge true \not\rightarrow \phi$$

  - constant outputs are minimal elements
- Least upper bound defined
- Greatest lower bound: sometimes defined
  - C.f. *shared refinement*

Main results (1)

- Refinement characterizes pluggability:
  - I' ≤ I **iff** for all environments E, pluggable(I,E) implies pluggable(I',E)
  - Note that this is **iff**
  - If we used an alternative notion of refinement (c.f., Meyer's subcontracting):

$$\forall X : in(\phi) \rightarrow in(\phi')$$
$$\forall X, Y : \phi' \rightarrow \phi$$

  - then **if** direction would not hold
  - neither would the last two examples

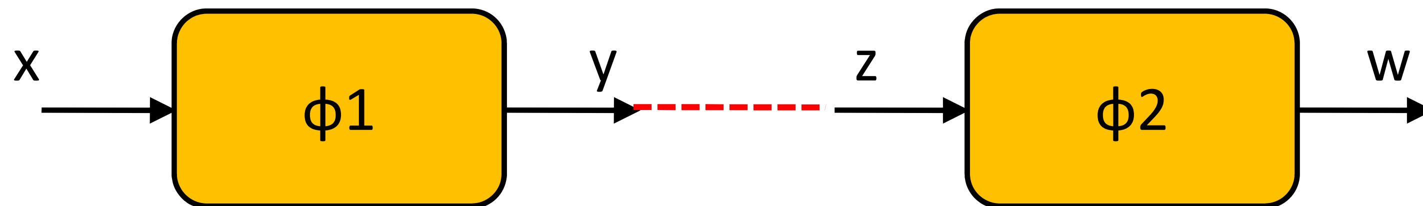
Plan of talk

- Relational interfaces
  - Stateless, stateful
- Environments and pluggability
- Refinement
  - Refinement and pluggability
- Composition
  - Connection, feedback
  - Preservation of refinement by composition

Composition: in a nutshell

- Composition by connection
- Composition by feedback
  - Arbitrary feedback not allowed:
    - It “breaks” the theory (refinement not preserved by feedback)
  - Restricted to Moore interfaces
    - Current outputs independent from some current inputs
  - Reasonable in most cases in practice
    - C.f., synchronous models like Simulink, Lustre, …

Composition by connection

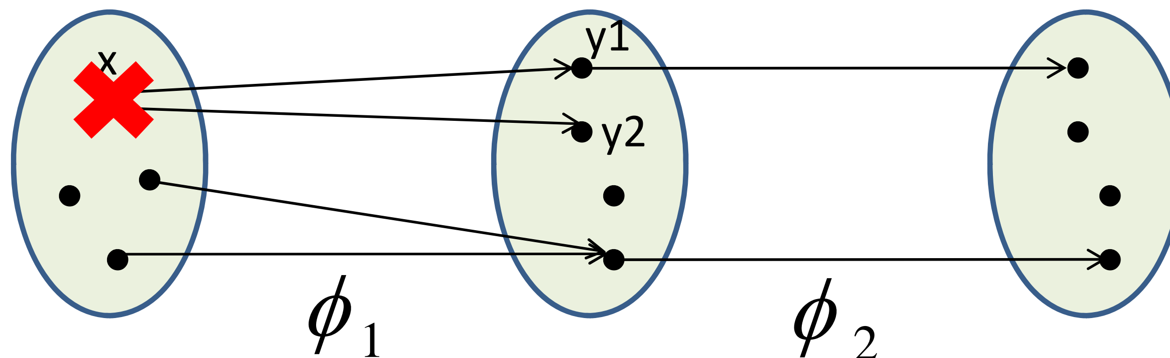


$$\phi := \phi_1 \wedge \phi_2 \wedge (y = z) \wedge \Phi$$

$$\Phi := \boxed{\forall y, z : \phi_1 \wedge (y = z) \rightarrow in(\phi_2)}$$

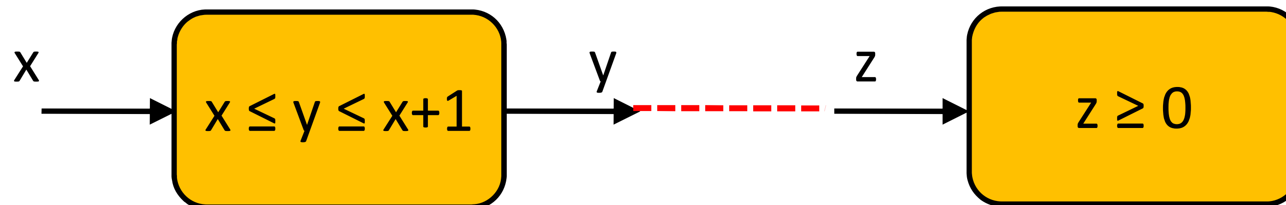
This is **not** composition of relations
(c.f., “demonic” vs. “angelic” non-determinism)

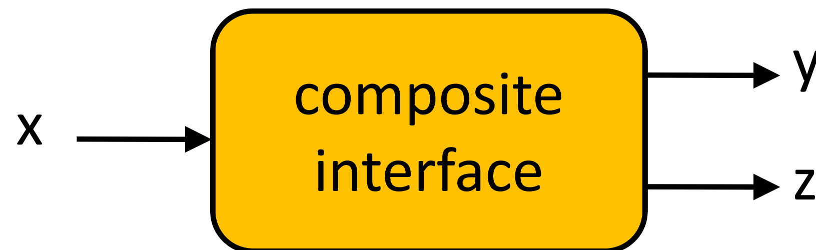
Composition by connection



This is **not** composition of relations
(c.f., “demonic” vs. “angelic” non-determinism)

Example of connection

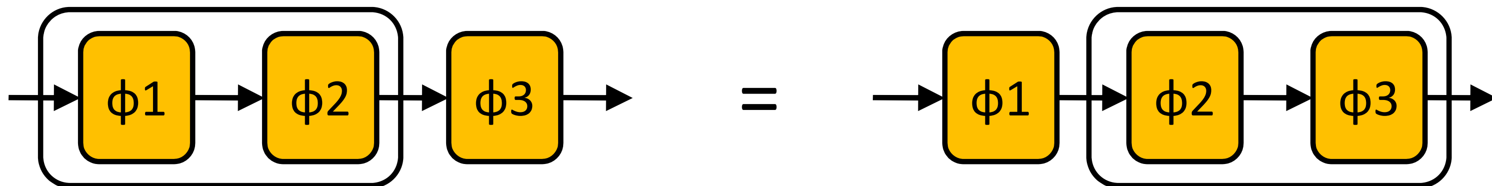




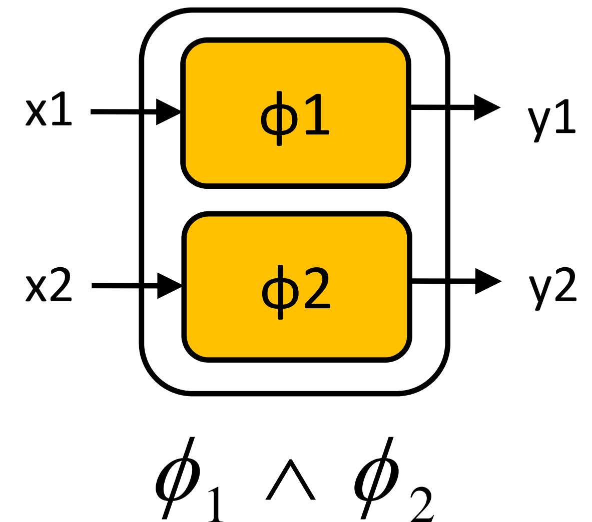
$$x \leq y \leq x+1 \wedge z \geq 0 \wedge y = z \wedge x \geq 0$$

Composition by connection

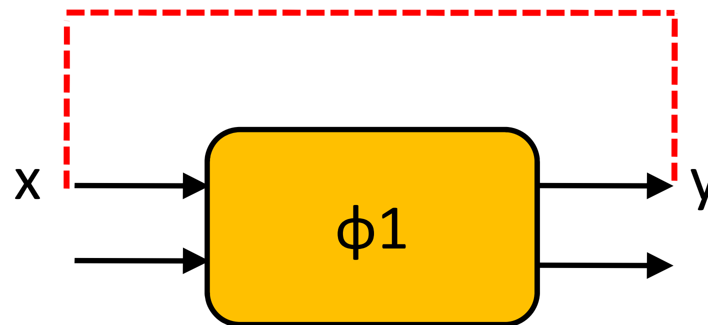
- Associative:



- **Parallel composition**
  - Special case = **empty** connection
  - Commutative



Composition by feedback



commutative

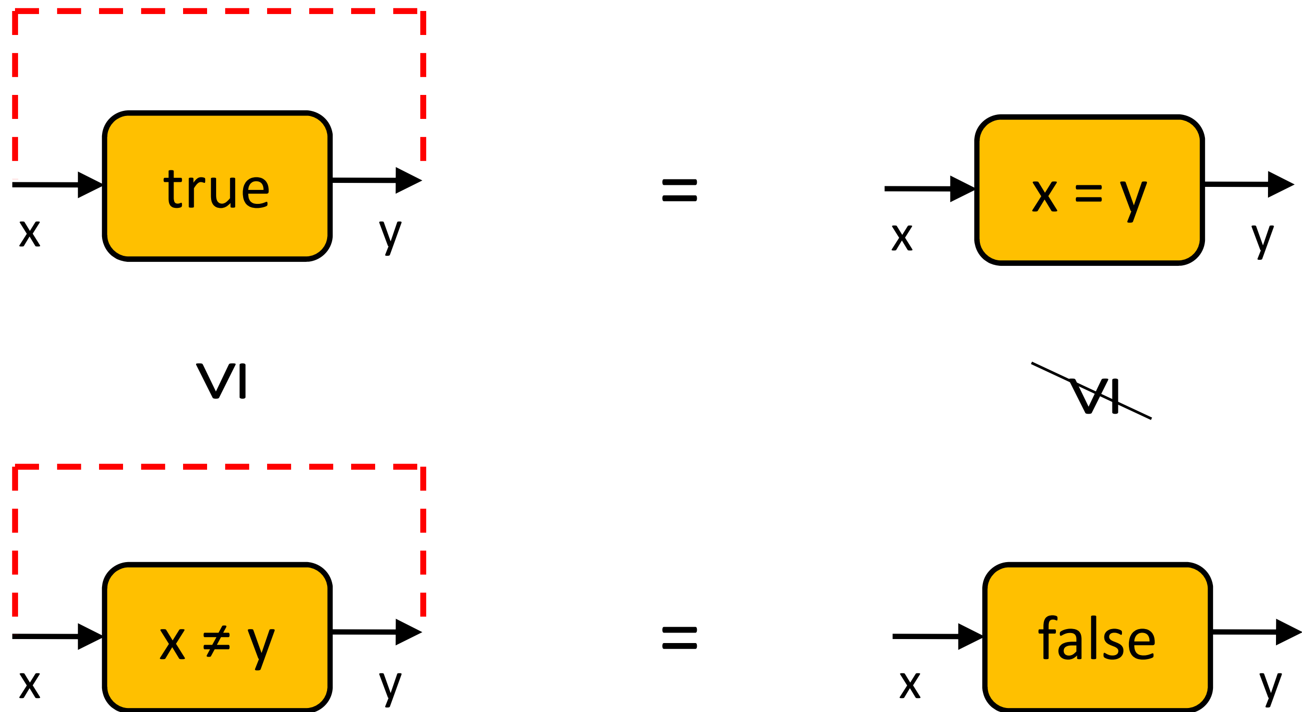
Interface must be Moore with respect to input x:
i.e., contract do not depend on x
(the Unit Delay is Moore)

$$\phi := \phi_1 \wedge (x = y)$$

Main results (2)

- Refinement preserved by composition:
  - If $A' \leq A$ and $B' \leq B$ then $\theta(A',B') \leq \theta(A,B)$
    - θ is a composition by connection
  - If $A' \leq A$ then $\kappa(A') \leq \kappa(A)$
    - κ is a composition by feedback
    - Both A and A' must be Moore
    - Refinement does not necessarily preserve Mooreness
      - E.g., $(y = 2x)$ refines $(y \bmod 2 = 0)$

Difficulties with arbitrary feedback



- Refinement would not be preserved by feedback …

Additional topics

see [EMSOFT'09] for details

- Hiding: removes output variables
  - Existential quantification for stateless interfaces
  - A bit trickier for stateful interfaces
- Shared refinement [Doyen et al 2008, Benveniste et al]
  - An interface that refines multiple others
  - Not always possible
- Input-complete interfaces

Input-complete interfaces

- All inputs are legal at all states:

$$in(\phi) \equiv true$$

$$y \neq 0 \rightarrow z = \frac{x}{y} \quad \text{vs.} \quad y \neq 0 \wedge z = \frac{x}{y}$$

Input-complete　　Non-input-complete

- Input-complete = receptive

Input-complete interfaces: theory becomes much simpler!

- Input-complete => well-formed
- Input-completion:
  - Input-complete version refines original

$$\phi' := \phi \vee \neg in(\phi)$$

- Connection, feedback, hiding preserve input-completeness
  - Connection is simplified:

$$\begin{cases} \phi := \phi_1 \wedge \phi_2 \wedge (y = z) \wedge \Phi \\ \Phi := \forall y, z : \phi_1 \wedge (y = z) \to in(\phi_2) \equiv true \end{cases}$$

- Refinement = implication
- Shared refinement = conjunction

so why do we need non-input-complete interfaces?

Why non-input-complete interfaces?

- Expressiveness:
  - Termination: some algorithms guarantee termination only if inputs satisfy some constraints => non-input-complete
    - Could model as input-complete with extra output in {T,?}, but need to handle this output during composition => comes to the same thing
- Flexibility in design:
  - Check local compatibility of interfaces: is their composition well-formed?
    - Catch errors earlier
  - Composition of input-complete is input-complete, which is always well-formed

Conclusions

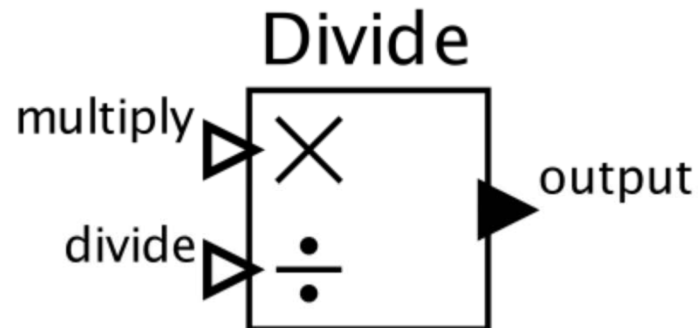
- Novel theory of relational interfaces
  - Generalizes previous attempts
  - Semantical, declarative, denotational, symbolic
  - Reasonable restrictions on feedback loops

- Main results:
  - Characterization of refinement by pluggability
  - Preservation of refinement by composition

On-going work

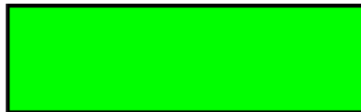
- Extend the theory
  - More flexibility in feedback:
    - Capture I/O dependencies in interfaces
    - A theory of fixed points for relations?
- Applications:
  - Case studies from the HW domain (circuits)
- Implementation in Ptolemy II
  - Only stateless interfaces for now

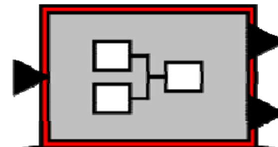
Director checks Interfaces



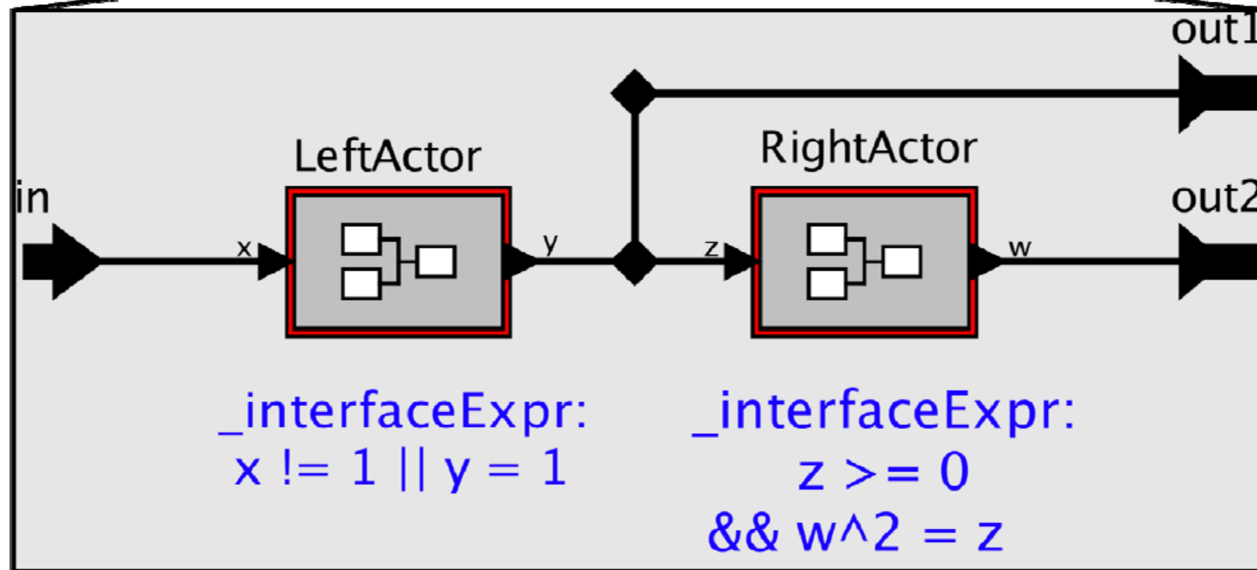
_interfaceExpr:
divide != 0
&& output == multiply / divide

Can infer composite interfaces

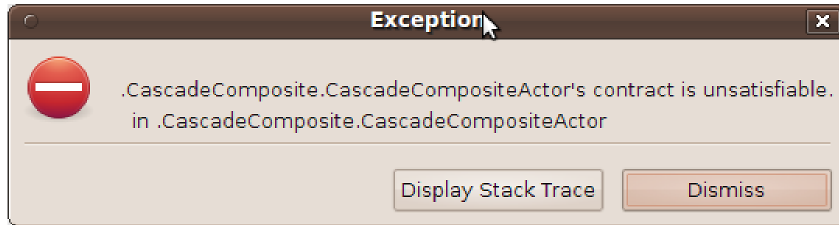


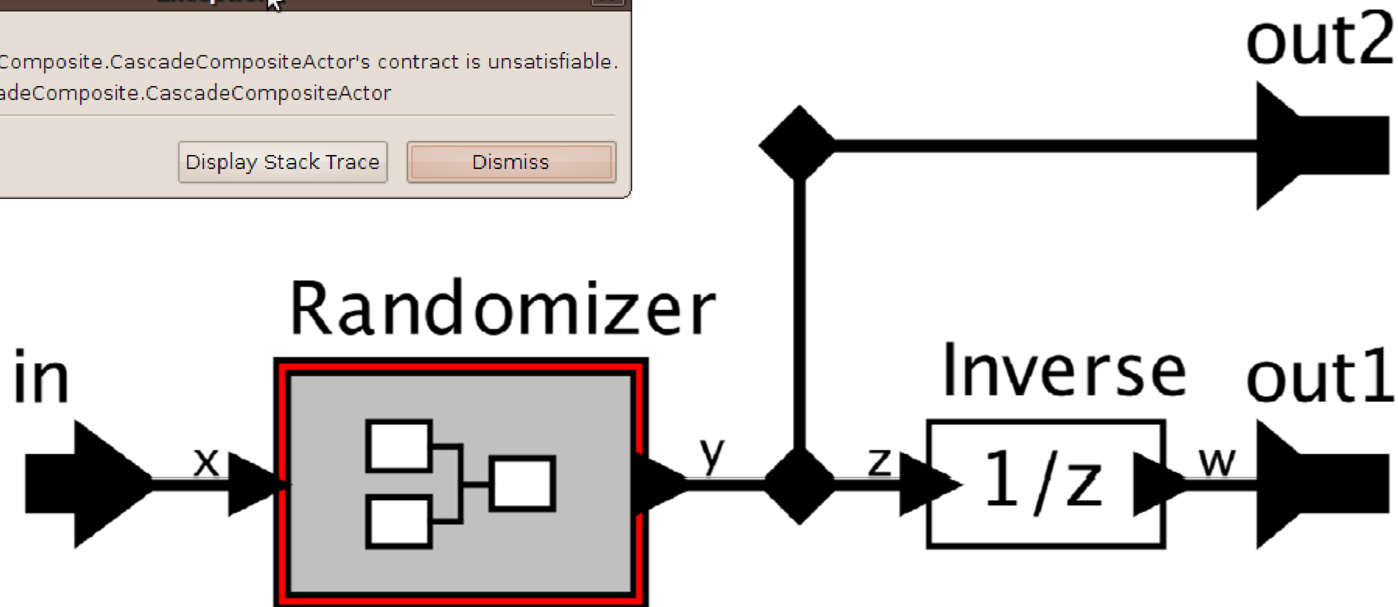


(and (= x in) (and (= z y) (and (>= z 0) (== (* w w) z)) (or (/= x 1) (= y 1)) (forall (y::int z::int) (=> (and (or (/= x 1) (= y 1)) (= z y)) (exists (w::int) (and (>= z 0) (== (* w w) z)))))) (= y out1) (= w out2) (= z out1))



Throws exception when composition impossible



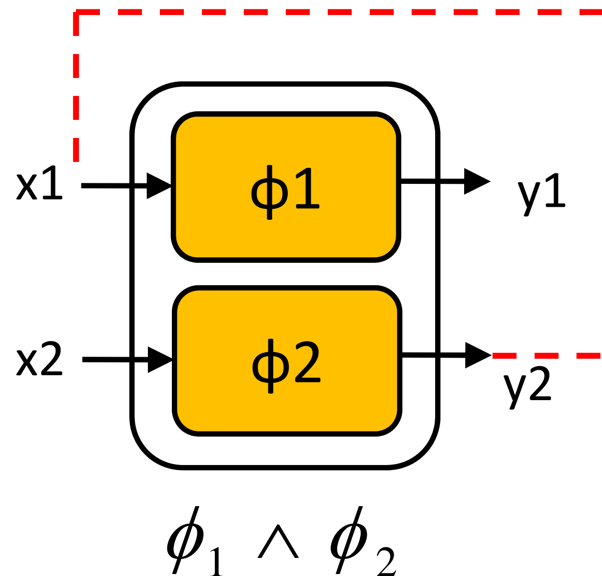


On-going work

- Extend the theory
  - More flexibility in feedback:
    - Capture I/O dependencies in interfaces
    - A theory of fixed points for relations?
- Applications:
  - Case studies from the HW domain (circuits)
- Implementation in Ptolemy II
  - Only stateless interfaces for now

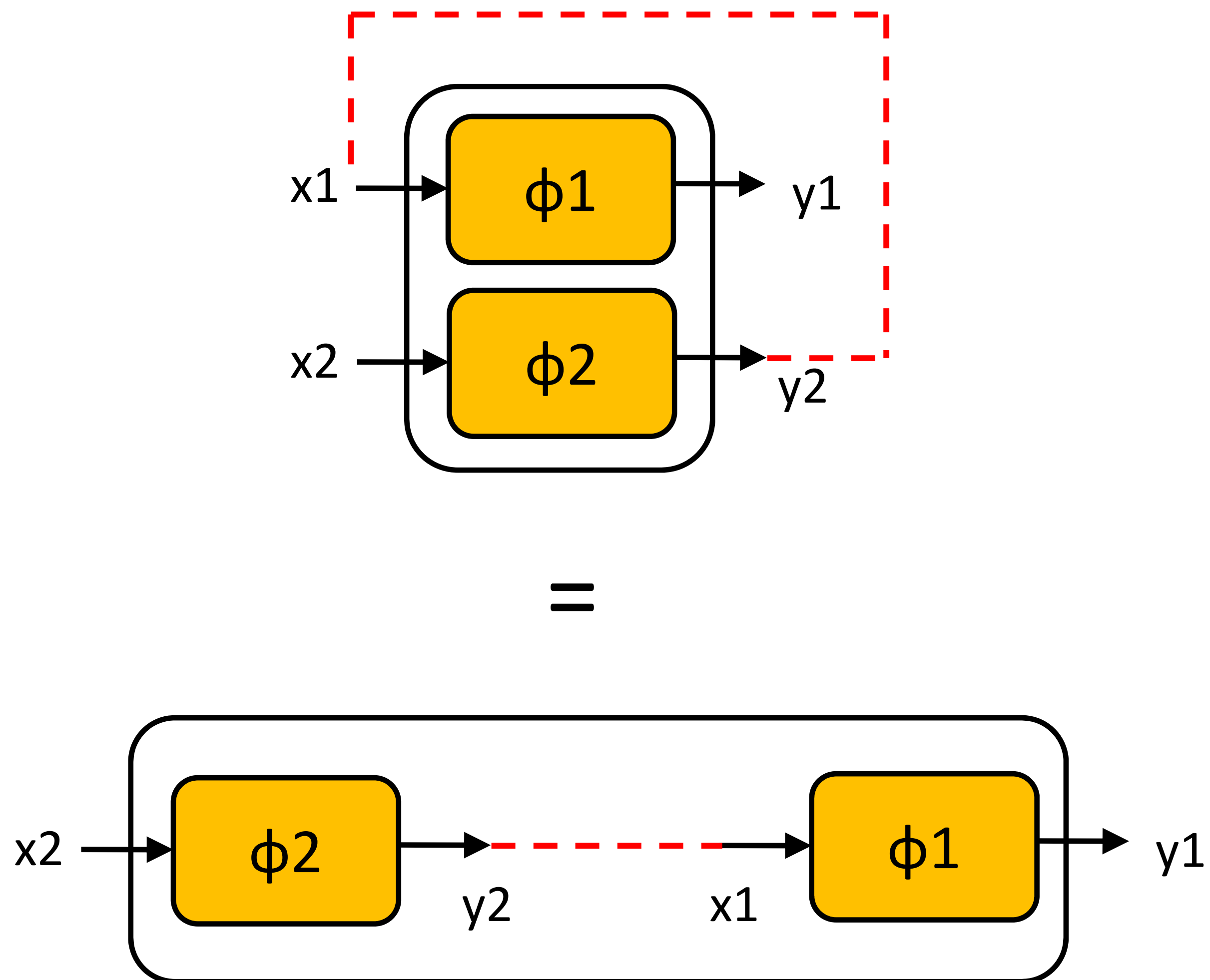
Limitations on feedback

- Consider the parallel composition of two interfaces:



- Need a way to capture I/O dependency information

Limitations on feedback



Thank you

- Questions?