An Extensible Type System for Component-Based Design

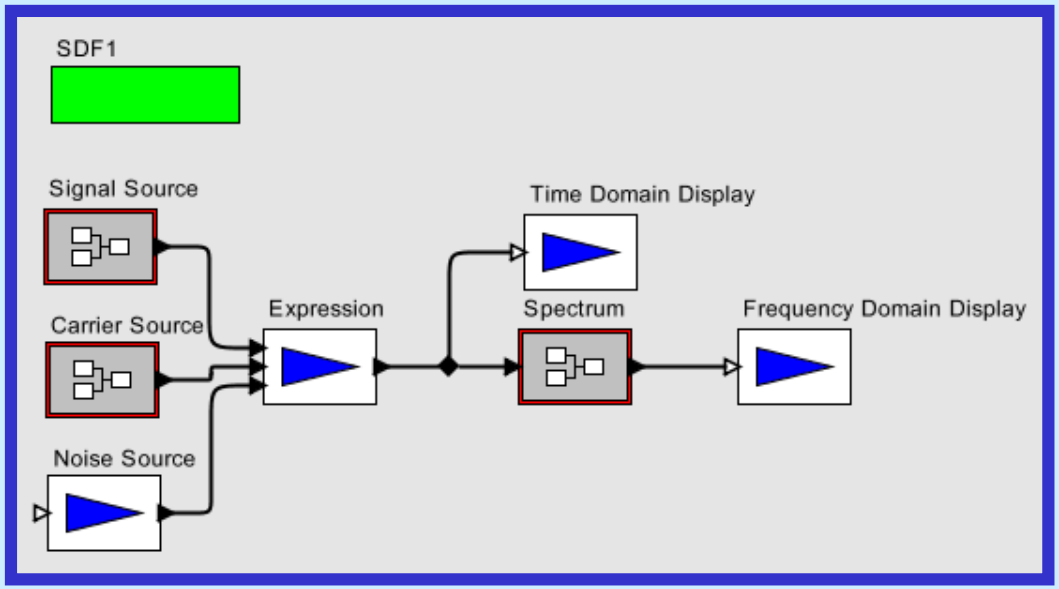
Yuhong Xiong
Advisor: Professor Edward A. Lee

Department of Electrical Engineering and Computer Sciences
University of California at Berkeley

DAC Ph.D. Forum, 6/11/02

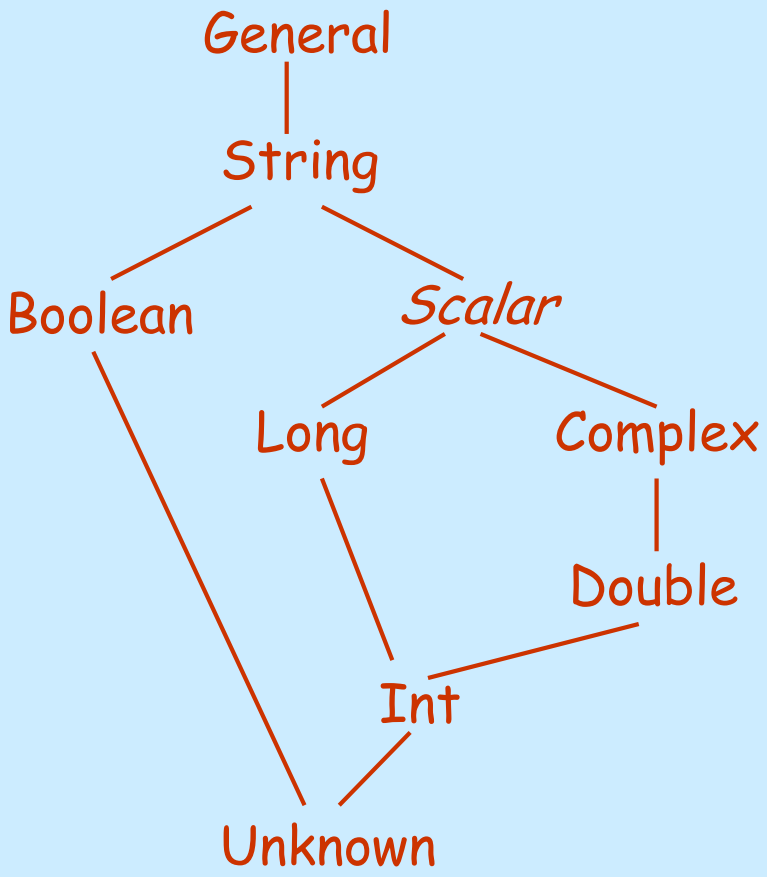
Component-Based Design

- Good for designing complex, concurrent, heterogeneous systems
- Two levels of interface:
  - data types and
  - dynamic interaction: communication & execution
- We propose a type system to address the constraints at these two levels



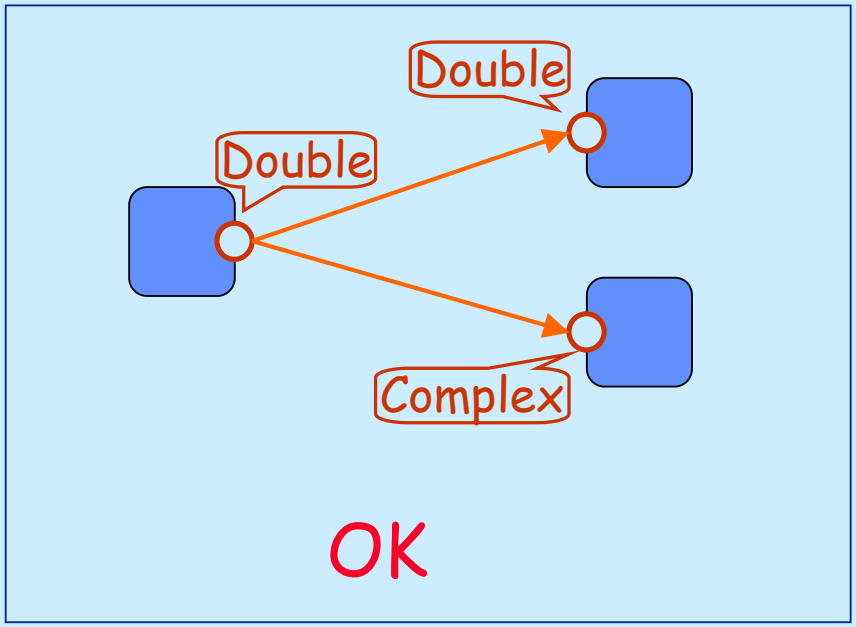
Data Type Lattice

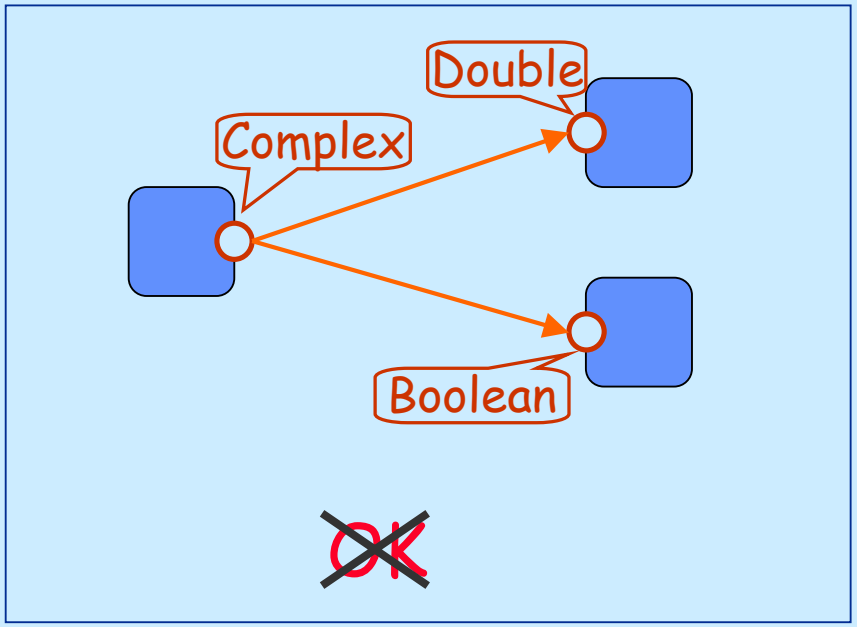
- Organize all types in a lattice structure
- This example lattice specifies lossless type conversion relation



Type Compatibility Rule

sendType ≤ receiveType

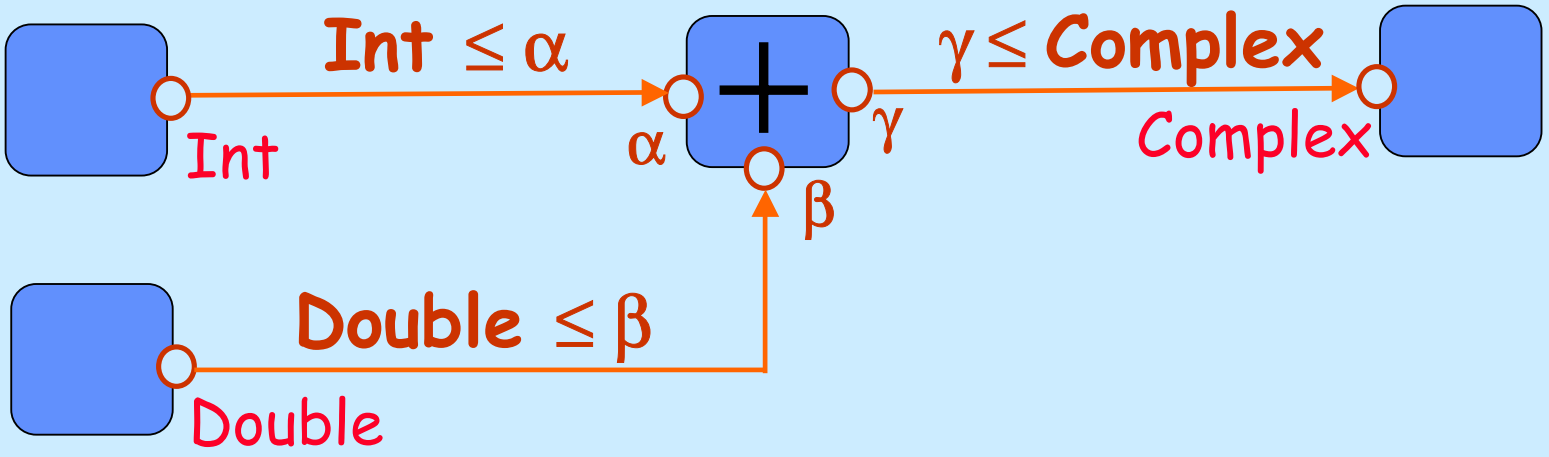




- Static type checking
- Type conversion

Type Constraints

$\alpha \leq \gamma$
$\beta \leq \gamma$
$\gamma \leq$ Complex



Efficient algorithm (Rehof & Mogensen) can find least solution

Structured Types
(arrays and records)

Goals:

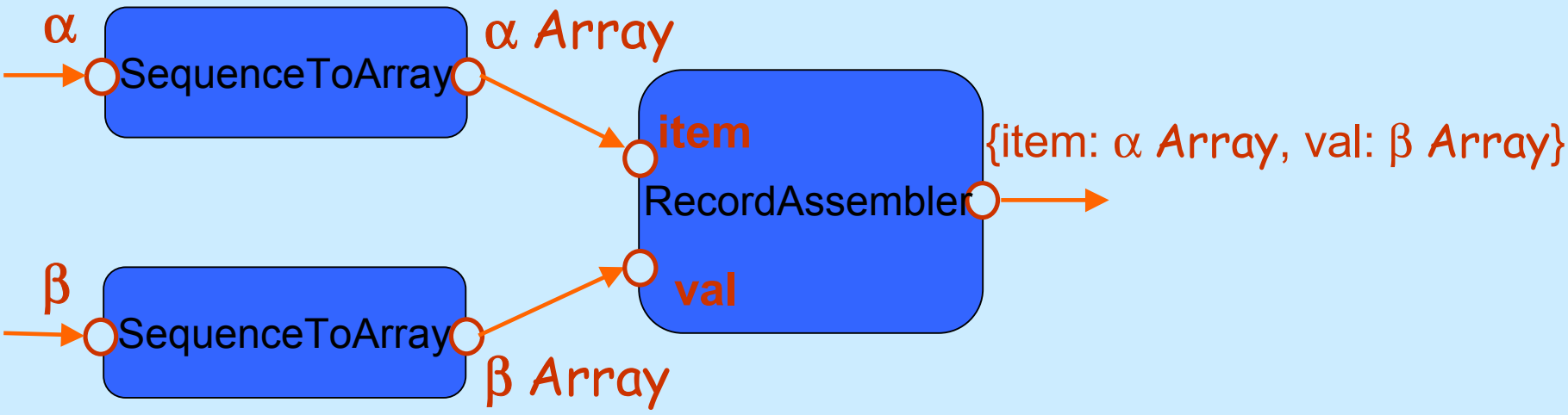
- Arbitrary element types. E.g. (int)array, ((int)array)array, array of records, records containing arrays, …
- Type constraints between element types and the types of other objects in system

Questions:

- Order relation among structured types?
- Structured types admitted by the inequality solving algorithm?
- Convergence on infinite lattice?

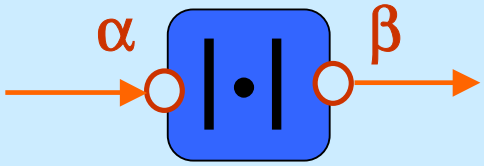
Actors Manipulating Structured Types

- SequenceToArray
- ArrayToSequence
- ArrayAppend
- ArrayElement
- ArrayExtract
- ArrayLength
- RecordAssembler
- RecordDisassembler
- RecordUpdater



Monotonic Functions in Type Constraints

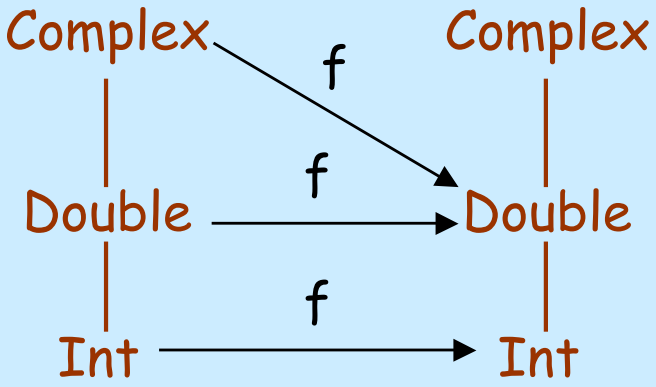
- Example: AbsoluteValue Actor
  - Works for Int, Long, Fix, Double, Complex
  - Output type is the same as the input, unless input is Complex
  - Output type is Double when input is Complex



$f(\alpha) \leq \beta$, where

$f(\alpha) = \text{Double}$ if $\alpha = \text{Complex}$

$= \alpha$ otherwise

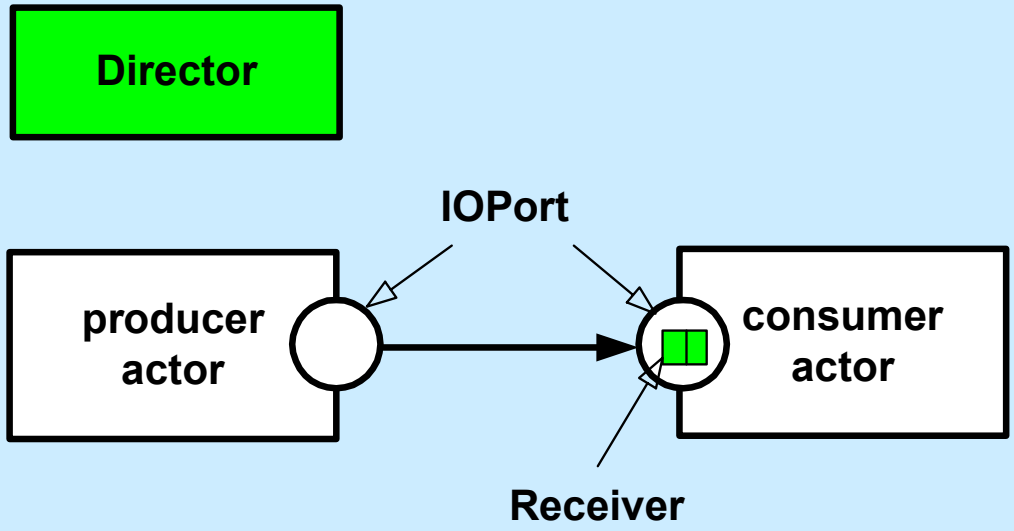


Behavioral Type

- Data types only specify static aspects of interface
- Proposal:
  - Capture the dynamic interaction of components in types
  - Use interface automata (de Alfaro & Henzinger)
  - Obtain benefits analogous to data typing
  - Call the result *behavioral types*
- Experimental platform: Ptolemy II

Interaction Semantics

- Flow of control issues
  - in Ptolemy II, these are defined by a *Director* class
- Communication between components
  - in Ptolemy II, this is defined by a *Receiver* class



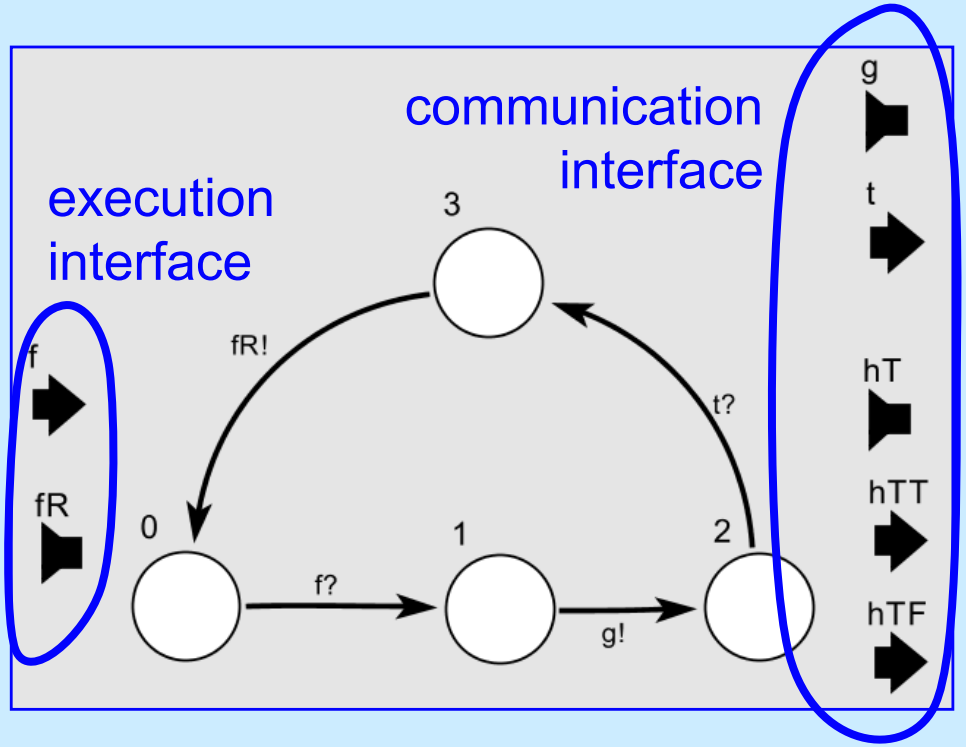
Actor interface for execution: fire

Receiver interface for communication: put, get, hasToken

Models of Computation

- Define the interaction semantics
- Implemented in Ptolemy II by a *domain*
  - Receiver + Director
- Examples:
  - Communicating Sequential Processes (CSP):
    rendezvous-style communication
  - Process Networks (PN):
    asynchronous communication
  - Synchronous Data Flow (SDF):
    stream-based communication, statically scheduled
  - Discrete Event (DE):
    event-based communication
  - Synchronous/Reactive (SR):
    synchronous, fixed point semantics

Example: Synchronous Dataflow (SDF) Consumer Actor Type Definition



Such actors are passive, and assume that input is available when they fire.

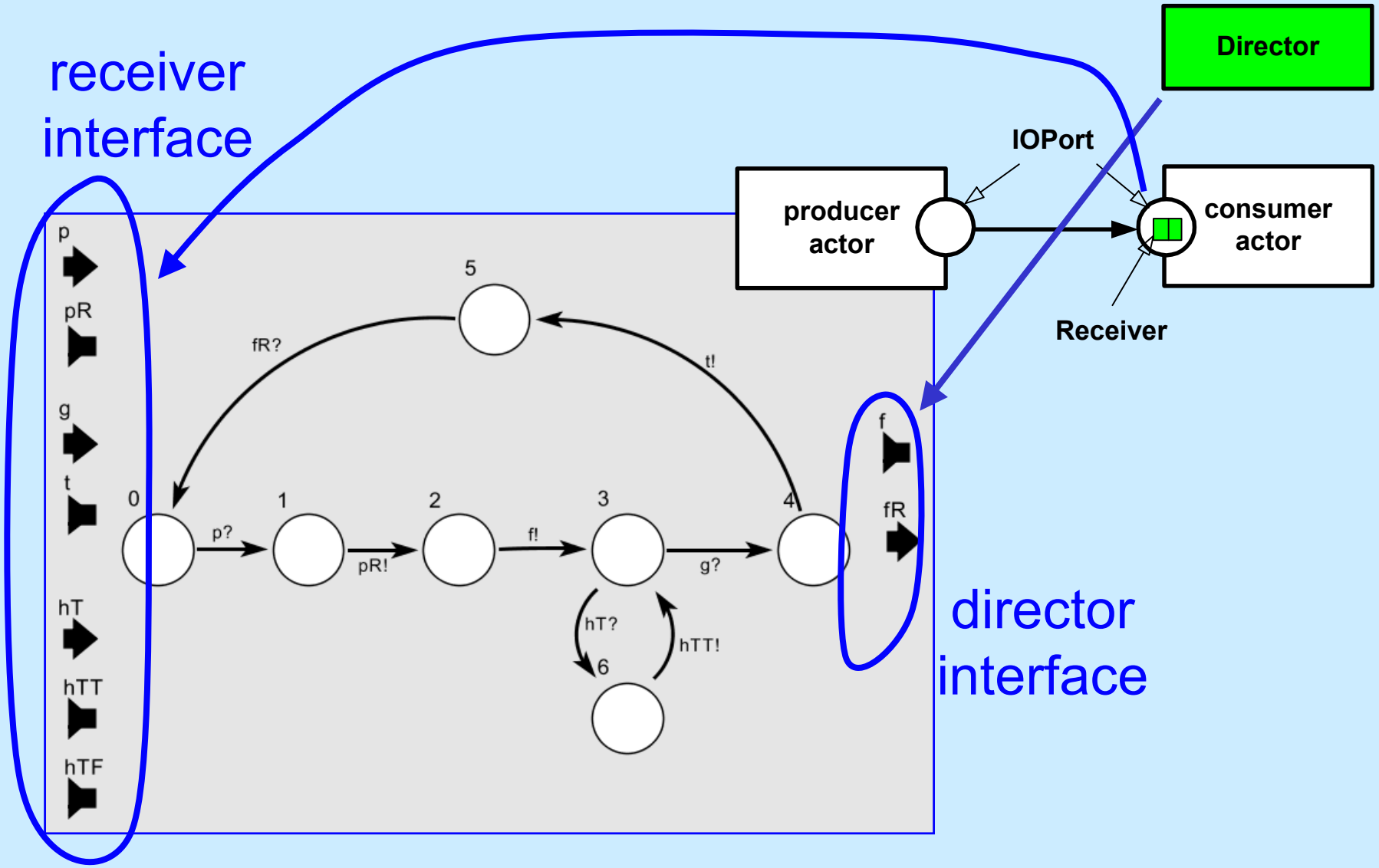
Inputs:

f	fire
t	Token
hTT	Return True from hasToken
hTF	Return False from hasToken

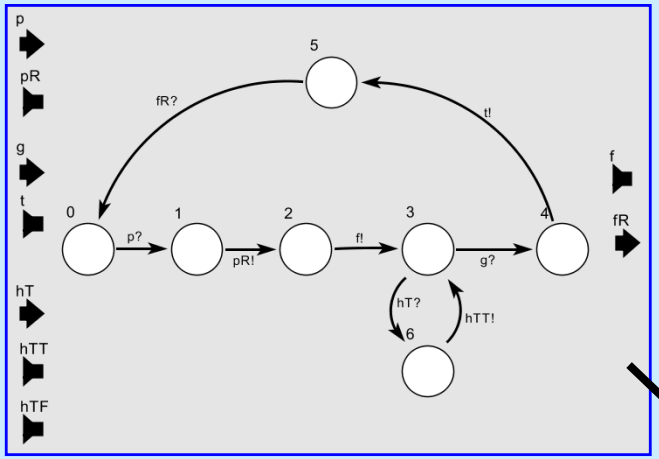
Outputs:

fR	Return from fire
g	get
hT	hasToken

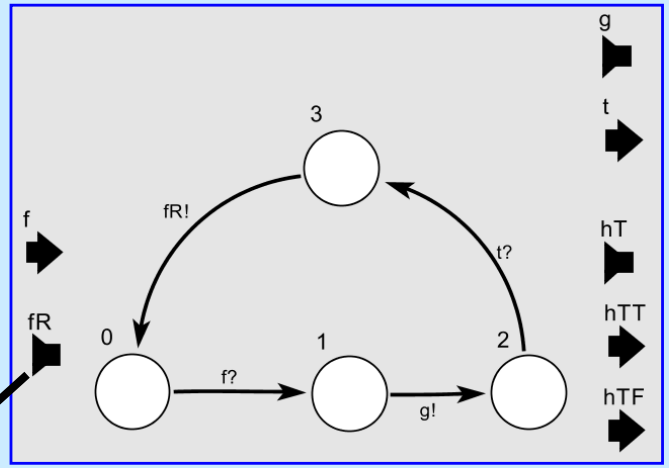
Type Definition - Synchronous Dataflow (SDF) Domain



Type Checking - Compose SDF Consumer Actor with SDF Domain

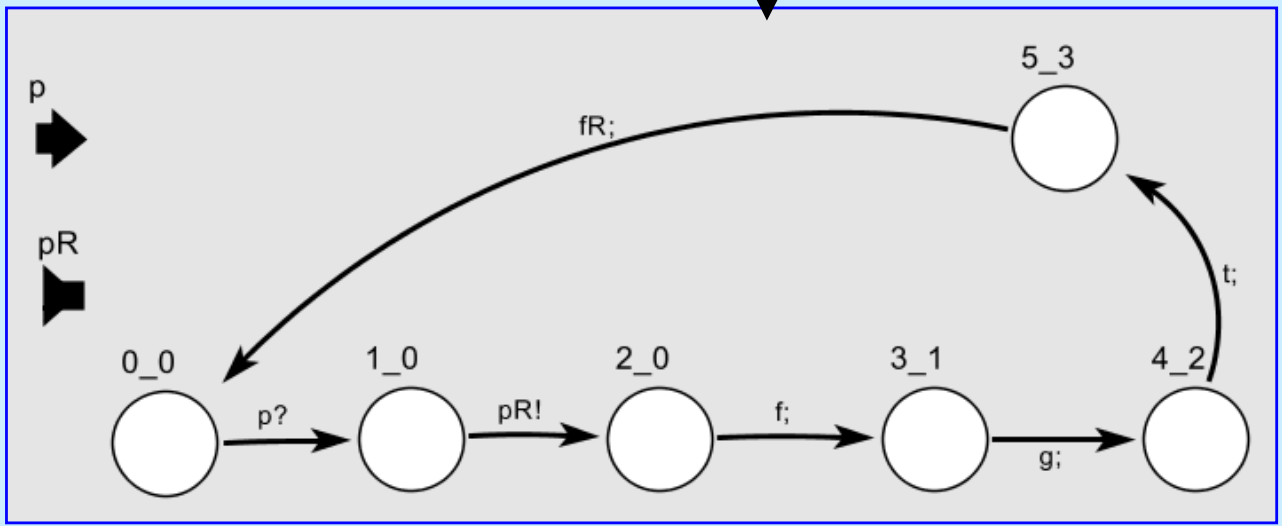


SDF Domain

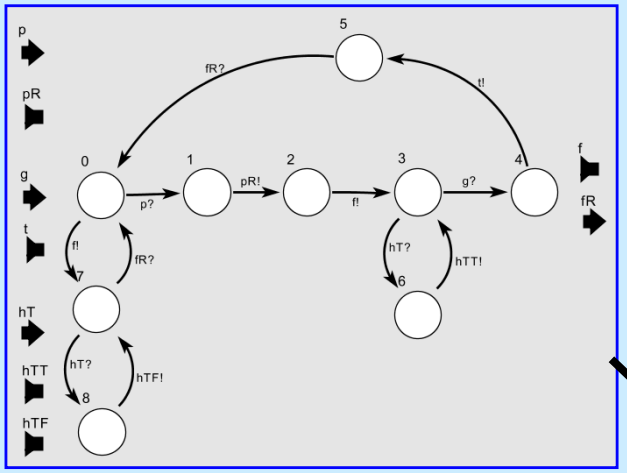


SDF Consumer Actor

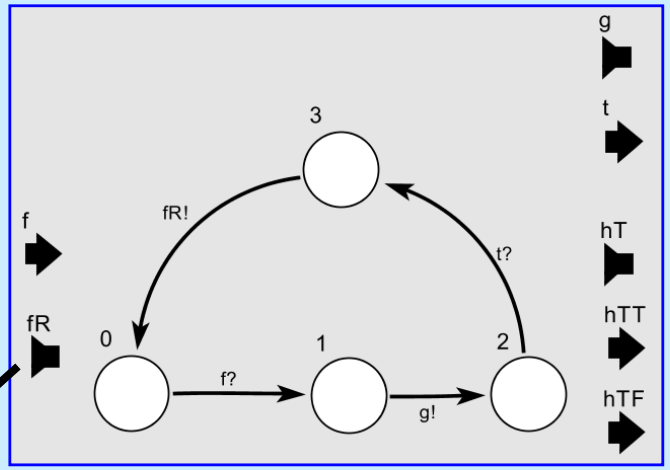
Compose



Type Checking – Compose SDF Consumer Actor with DE Domain



DE Domain



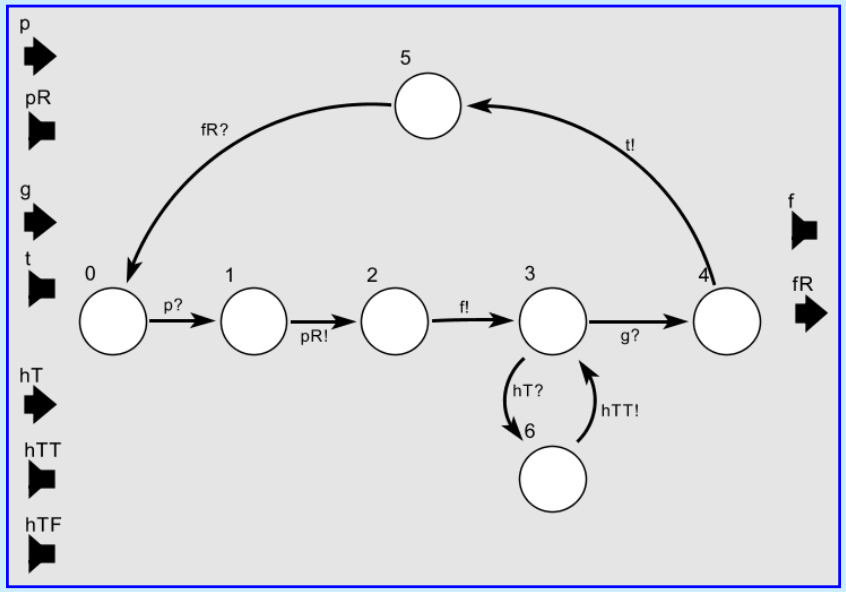
SDF Consumer Actor

Compose

- Empty automaton indicates incompatibility
- Composition type has no behaviors

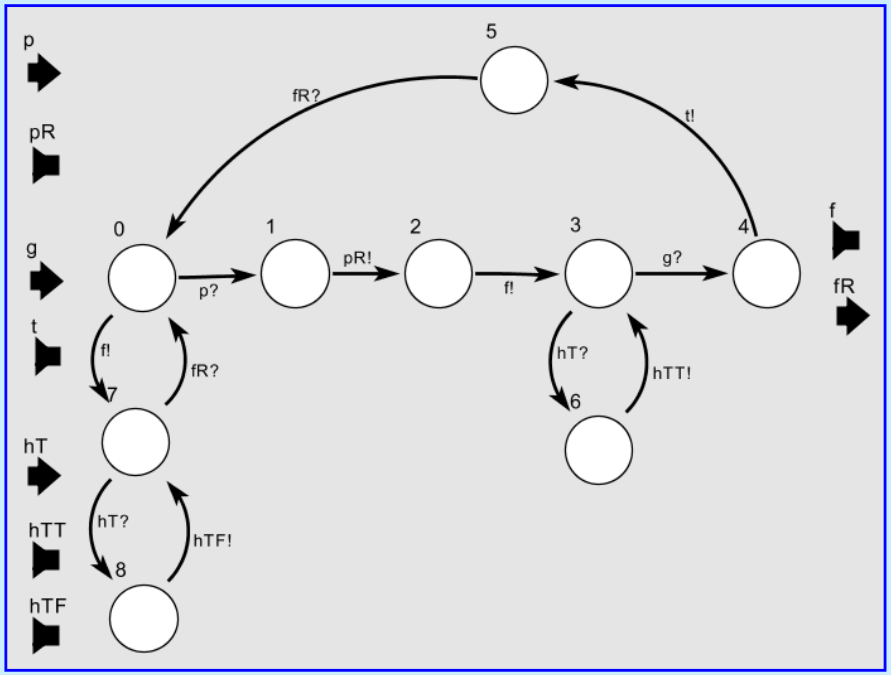
Subtyping Relation
Alternating Simulation: SDF ≤ DE

SDF Domain

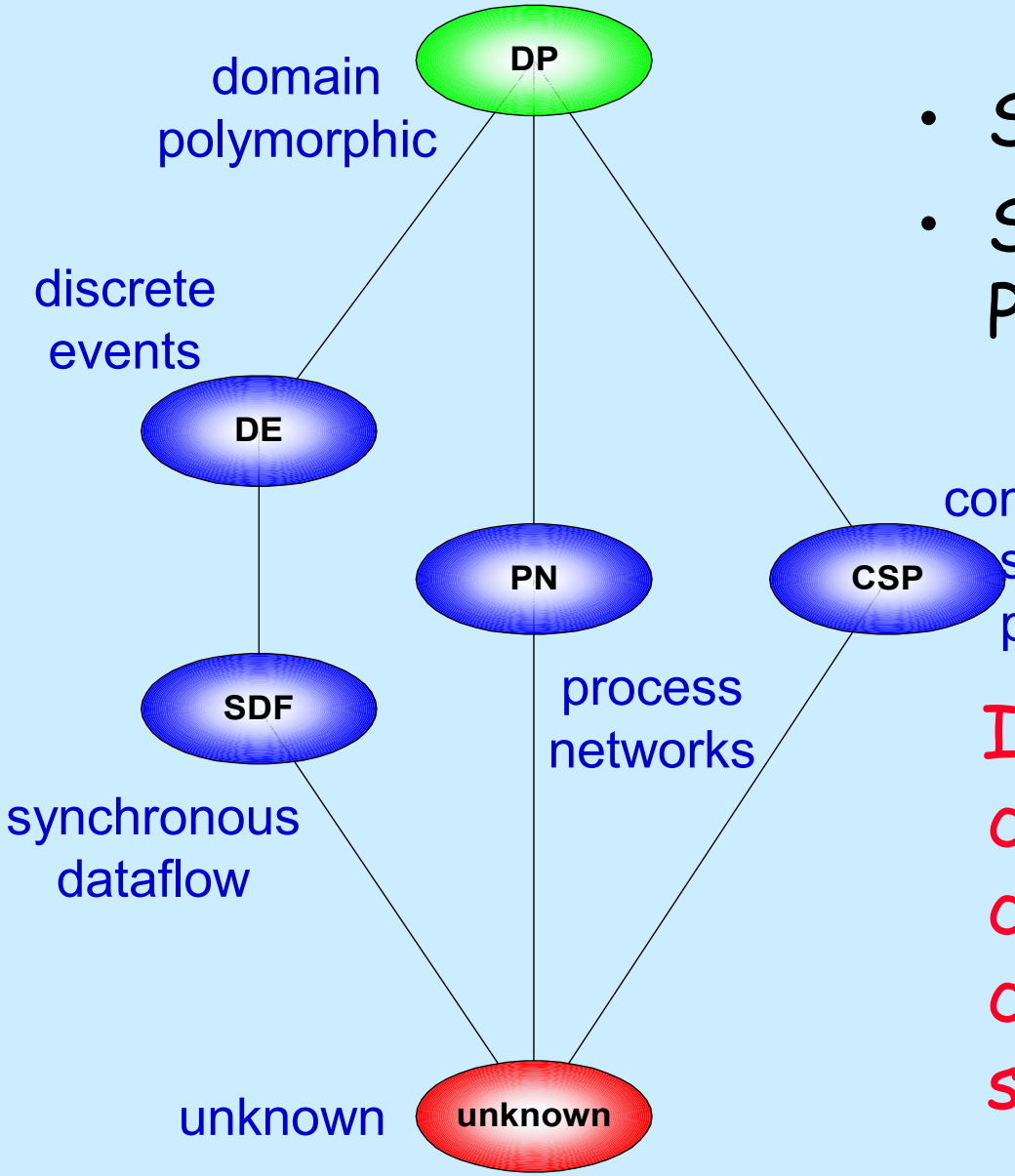


≤

DE Domain



Behavior-Level Type Lattice – Defined by Alternating Simulation



- Subtyping relation
- Shown here for a few Ptolemy II domains

If an actor is compatible with a certain type, it is also compatible with the subtypes