Modular Reasoning for Actor Specification Diagrams

Scott F. Smith
The Johns Hopkins University

Carolyn L. Talcott
Stanford University

February 17, 1999

for FMOODS ’99

Reasoning About Open Systems Project

- Collaboration with Agha, Mason, Smith, **Talcott**
- Rigorous reasoning for open distributed systems
- General multi-language framework
- General with respect to data
- Proof principles
- Applicability to real examples

This talk: a new graphical language for high-level specification

Language Design Goals

A language for specifying message-passing behavior that is

- Expressive
- Intuitively understandable by non-experts
- With a rigorous underlying semantics

Choice is a *graphical format* for ease of communication

Our approach

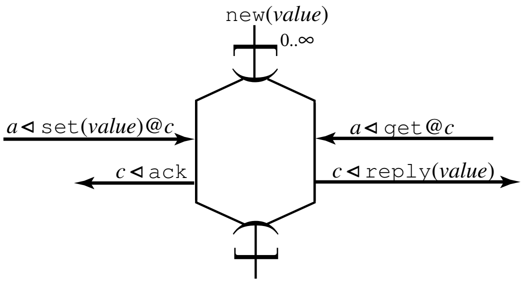
UML sequence diagram style with

- Significantly greater expressivity
- Usefulness across a wider portion of the design cycle (not just in initial design phases)
- Rigorous underpinnings
- Algebra of composition, restriction
- Elements of programming logic added

A peek at an example

This simple cell holds a single value, which responds to `set` and `get` messages.

Cell(a) =



Outline of the talk

1. Actor communication basics
2. Diagram syntax
3. Examples
4. Actor Theory framework
5. Operational semantics of diagrams
6. Example proofs of properties: function composer
7. Conclusions and Future Work

Actor Communication Basics

- Actors each have a unique *name*, $\boxed{a}$
- Actors may dynamically create other actors
- Actors only communicate by passing messages, $\boxed{a \triangleleft M}$
  – a is destination, M is data
- Acquaintance function, $\boxed{acq(M)}$
  – the actor names communicated in a message M
- Messages are sent asynchronously
- All messages must eventually arrive (fair delivery)

Open Systems Modeling

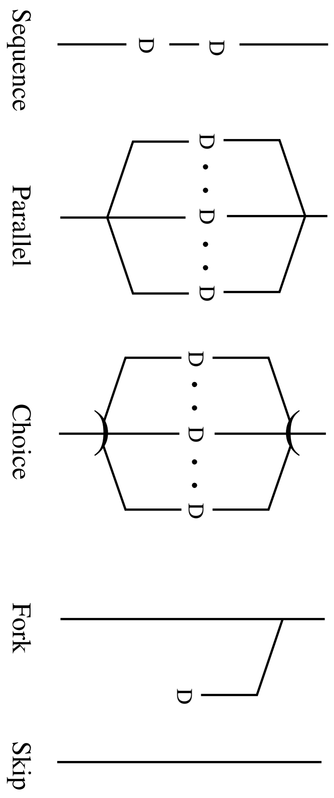
- System is open, interacting with (arbitrary) environment
- *External actors* $a \in \chi$ are interacting outsiders
- *Receptionists* $a \in \rho$ are locals interacting with outsiders
- Sets χ and ρ evolve over time

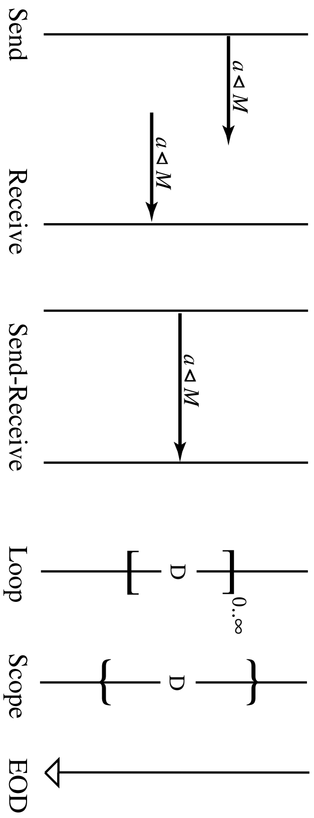
Interaction Path Model

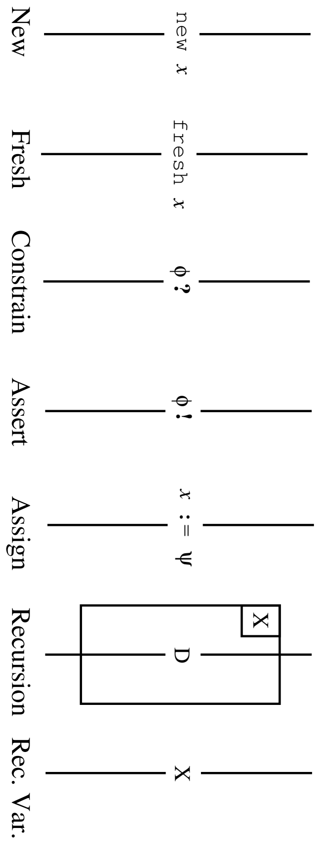
- $\texttt{in}(a \triangleleft M)$ is an input action
  —data arriving from environment; $a \in \rho$
- $\texttt{out}(a \triangleleft M)$ is an output action
  —data sent to environment; $a \in \chi$
- An actor system “run” is a sequence of $\texttt{in}/\texttt{out}$ actions
- Each such sequence is an *interaction path*
- Actor systems modelled by their set of interaction paths

—The model is a trace-style model but is semantically clean, unlike CSP traces.

Diagram Syntax







Ancestry of Features

Feature	Source
asynchrous messaging	actors
parallal and choice	process algebra
constrain and assert	Dijkstra program logic
cross-edge messaging	UML sequence diagrams
arbitrary math. universe	(programming logics)
state and assignment	(programming langauges)

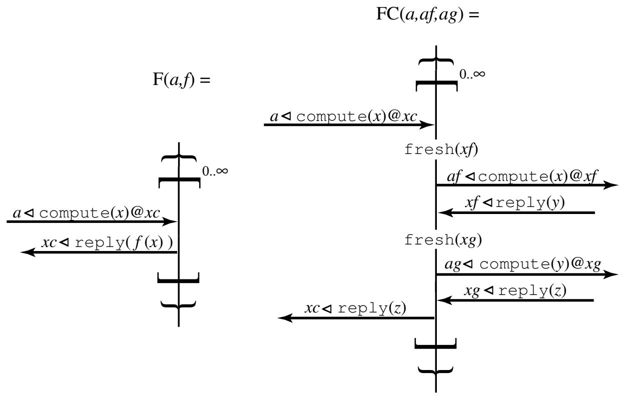
General points about the language

- Stateful; shared variables across threads possible
- Mathematical domain of discourse is not fixed but can be taken to be set theory
- A grammatical notation also exists (see paper)
- Some diagrams not realizable as actor programs
- Can encode standard constructs: if-then; while-do; synchronous messaging

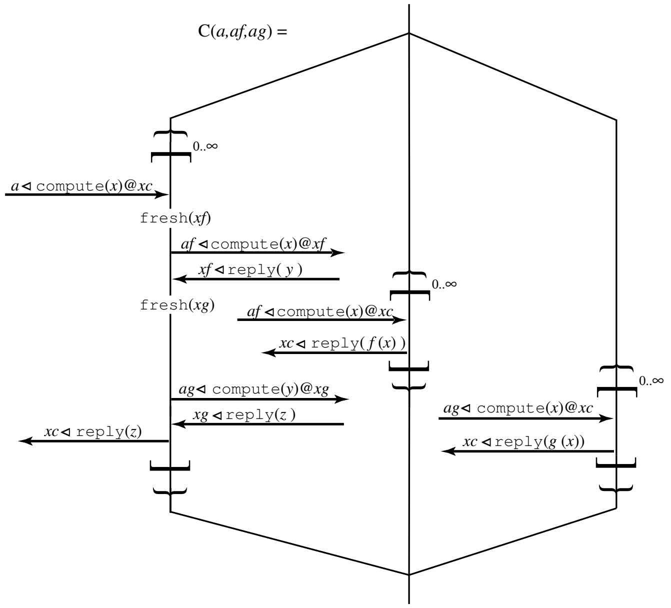
Function Composer—Components

A distributed method for computing $g \circ f$ for composable functions f and g. Components are F and FC

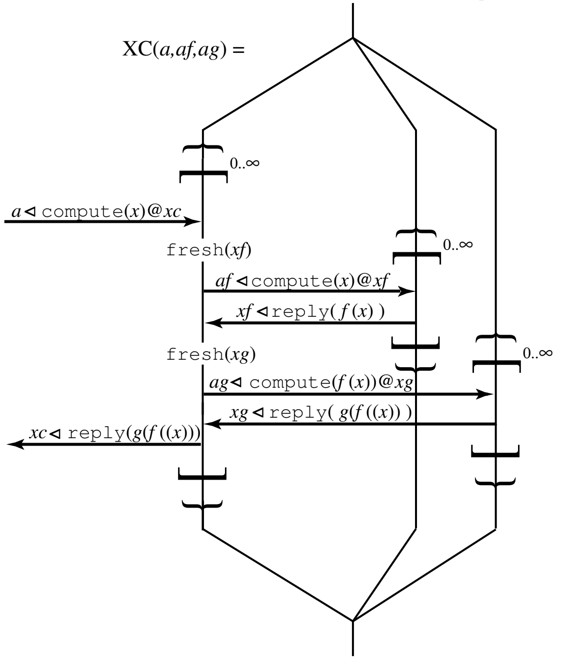
- F computes a function f
- FC composes two functions f and g



Function Composer—System



Refined Function Composer



Cross-edges assert sends and receives match up 1-1

Relating Specification Diagrams

Need useful notions of how implementation D_I satisfies specification D_S.

First Notion: full and faithful satisfaction of a specification.

Definition 1 (strong satisfaction):

$$\langle D_I\rangle^{\rho}_{\chi} \mathrel{|\!\!=\!\!|} \langle D_S\rangle^{\rho}_{\chi} \text{ iff}$$
$$[\![\langle D_I\rangle^{\rho}_{\chi}]\!] = [\![\langle D_S\rangle^{\rho}_{\chi}]\!]$$

where

- a *top-level* specification diagram includes an interface, notated $\langle D\rangle^{\rho}_{\chi}$
- $[\![\langle D\rangle^{\rho}_{\chi}]\!]$ is interaction path semantics of $\langle D\rangle^{\rho}_{\chi}$

Strong Satisfaction and the Function Composer

High-level specification for computing $g \circ f$ is $\mathrm{F}(a, g \circ f)$

Theorem 2:

$$\langle \mathrm{C}(a, f, g, af, ag)\rangle^{a}_{\emptyset} \mathrel{|\!\!=\!\!|}$$
$$\langle \mathrm{XC}(a, f, g, af, ag)\rangle^{a}_{\emptyset} \mathrel{|\!\!=\!\!|}$$
$$\langle \mathrm{F}(a, g \circ f)\rangle^{a}_{\emptyset}$$

Proof will be sketched later in talk.

Asserting Properties of Specifications Diagrammatically

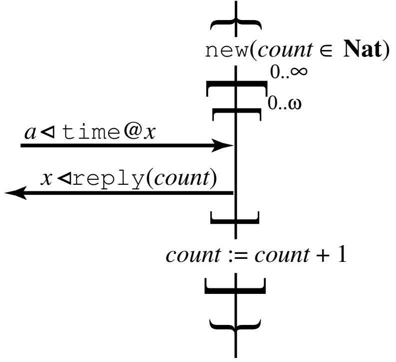
- Safety and liveness properties can be asserted directly in the specification diagram language.
- The ability to express assertions diagrammatically means there is less need to learn a specialized logic in which assertions are written.
- More practical possibility of getting engineers to use.

Three techniques for asserting properties now covered

Running Example: Ticker

A Ticker is a monotonically increasing counter

$\mathrm{Ticker}(a) =$



- Finite Inner loop $0 \ldots \omega$ guarantees progress of *count*.
- A top-level ticker: $\langle \mathrm{Ticker}(a) \rangle_{\emptyset}^{a}$.

Assertions I - Loose Satisfaction

Loose satisfaction is a standard notion of satisfaction:

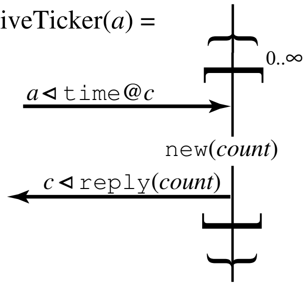
$$\langle D_I \rangle_{\chi}^{\rho} \models \langle D_S \rangle_{\chi}^{\rho} \quad \text{iff} \quad [\![\langle D_I \rangle_{\chi}^{\rho}]\!] \subseteq [\![\langle D_S \rangle_{\chi}^{\rho}]\!].$$

“Diagram D' has property P_{D}” is expressed as

$$\langle D' \rangle_{\chi}^{\rho} \models \langle D \rangle_{\chi}^{\rho}$$

Consider for instance the $\mathrm{LiveTicker}(a)$

$\mathrm{LiveTicker}(a) =$



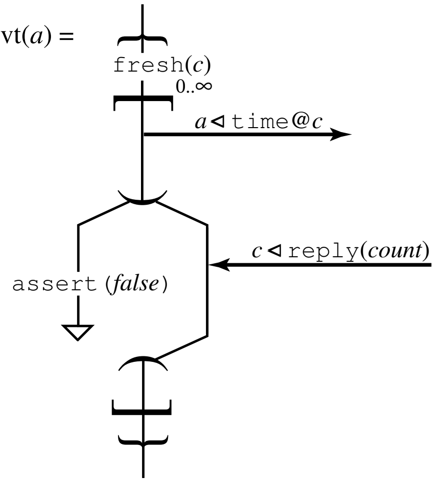
Assert: $\langle \mathrm{Ticker}(a) \rangle_{\emptyset}^{a} \models \langle \mathrm{LiveTicker}(a) \rangle_{\emptyset}^{a}$

– all `time` messages sent to the Ticker will receive a reply

Assertions II - Environment-Based Assertions

Specify an environment which *fails* when desired property fails.

$\mathrm{LiveTickerEnvt}(a) =$

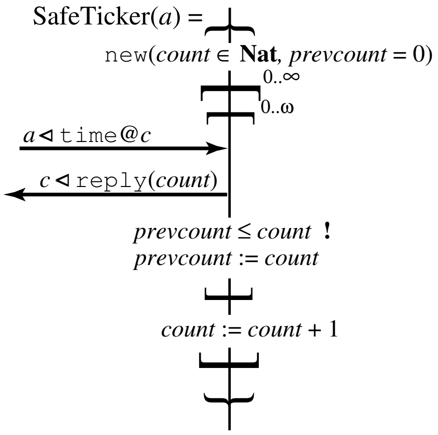


Assert: $\models \langle \mathrm{Ticker}(a) \mid \mathrm{LiveTickerEnvt}(a) \rangle_{\emptyset}^{\emptyset}$

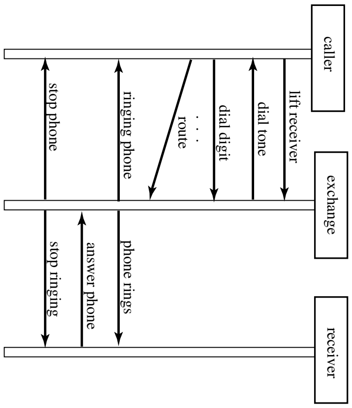
(Validity $\models \langle D \rangle_{\chi}^{\rho}$ means no `assert` fail.)

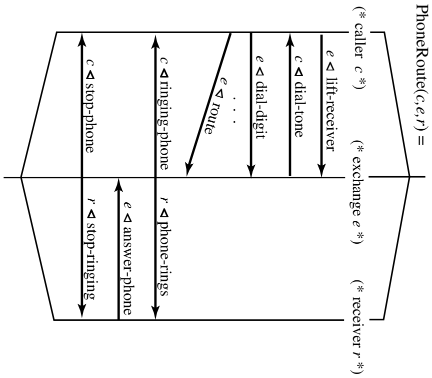
Assertions III - Safety Checks

Decorate a specification with assertions which must hold.



Assert: $\models \langle \text{SafeTicker}(a) \rangle^{a}_{\emptyset}$.





Actor Theories

A general semantic framework for actor systems

- abstracts from notational details
- enriches the basic actor computation model to express
  - synchronization between two or more actors
  - constraints on the environment

Actor theories can be used for

- semantics for programming and specification languages
- direct specification of actor system components
- relating actor system descriptions in different notations

Actor theory – Structure

An actor theory extends communication basics with

- States σ local state – acquaintances, script, . . .
- Reaction Rules $l : \sigma_0 \xrightarrow[\mu_s]{\mu_r} \sigma_1$
  - rule label l
  - source and target states σ_0, σ_1
  - received/consumed messages μ_r
  - sent/generated messages μ_s
- States and rules must obey the Actor Theory Laws
  - locality
  - parametricity in names

Actor theory configurations and transitions

- Configurations $K = \langle \sigma, \mu \rangle^{\rho}_{\chi}$
  - (ρ, χ) the interface of K
  - σ the internal state
  - μ the pool of pending messages
- Transitions $K \xrightarrow{tl} K'$
  - internal computation: $tl = l(fA, \mu_r, cA)$
  - input to a receptionists: $tl = \mathtt{in}(a \triangleleft M)$
  - output to an external actor: $tl = \mathtt{out}(a \triangleleft M)$
- Computations – infinite sequences of transitions

Interaction Semantics

The interaction semantics of a configuration, $[\![K]\!]$, is the set of interaction paths associated to the admissible computations of K

- each interaction path consists of an interface and a sequence of inputs and outputs
- the interaction path associated to a computation, $\mathit{cp2ip}(\pi)$, has
  - the same interface as the inital configuration
  - i/o sequence the subsequence of i/o labels of the computation
- $[\![K]\!] = \{\mathit{cp2ip}(\pi) \mid \pi \in \mathcal{A}(K)\}$

Specification Diagram semantics

- States which are diagrams (slightly enriched)
- Rules which traverse diagrams
  - interleaving parallel threads
  - unfolding recursive diagrams
  - updating state
  - sending and receiving messages
  - checking constraints
- Admissibility annotations – receives are mandatory

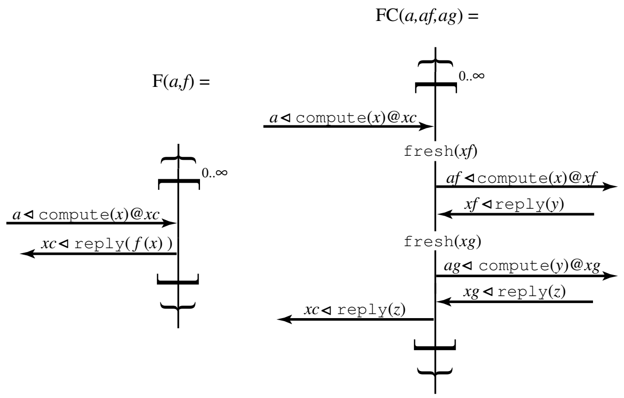
Actor theory toolkit

- Message restriction – a global disabling
- State restriction – focus attention
- Sum and Product operations
- Big-Step Transformation
  - groups sequences of internal transitions
  - reduces interleavings
- Message internalization
- Specialization – combines state and message restriction, internalization, and big step.

The Function composer example - I

Recall the composition of the function composer and two function computers:

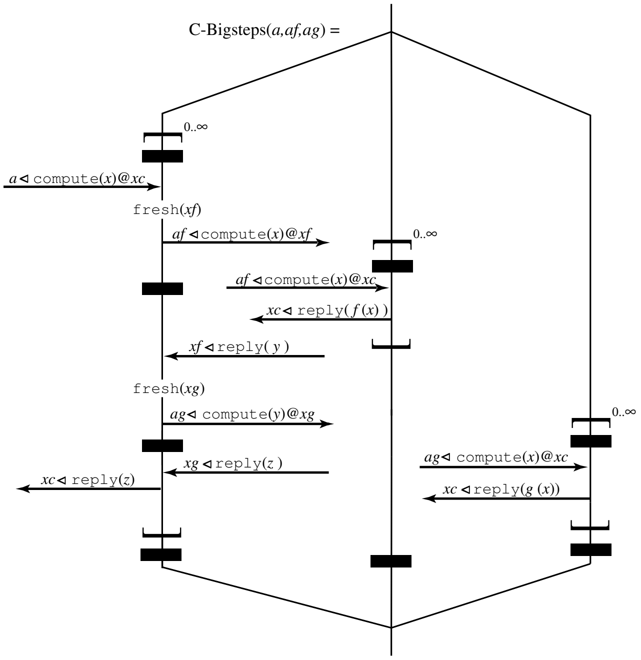
$$\mathrm{C}(a, f, g) = (\mathrm{FC}(a, af, ag) \mid \mathrm{F}(af, f) \mid \mathrm{F}(ag, g))$$



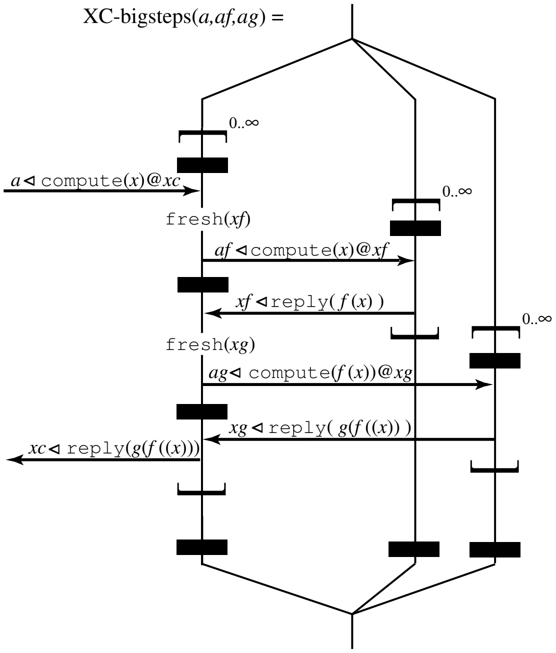
Theorem:

$$\langle \mathrm{C}(a, f, g, af, ag)\rangle^{a}_{\emptyset} \;\mathrel{|\!\!=\!\!|}\; \langle \mathrm{F}(a, g \circ f)\rangle^{a}_{\emptyset}$$

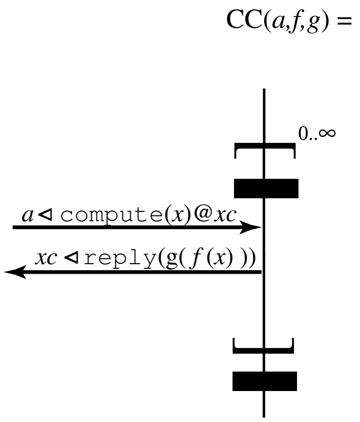
The Function composer example - II



The Function composer example - III



The Function composer example - IV



Future work

- Test on ever larger examples

- Rigorously develop graphical version of transformations

- Formalize how diagrams assert properties

- Add real-time constraints

- A more realistic version with an implemented diagram layout tool