Observability, Classes, and Object Connectivity

Erika Ábrahám   Marcello Bonsangue   Frank S. de Boer   Martin Steffen

Christian-Albrechts University Kiel

Summer Research Institute, EPFL Lausanne
July 2004

,

Structure

Full abstraction: starting point

- basically: comparison between 2 semantics, resp. 2 implied notions of equality
- given a reference semantics, the 2nd one is
  - neither too abstract = sound
  - nor too concrete = complete
- Milner [4], Plotkin [7] for λ-calculus/LCF
- various *variations* of the theme

Full abstraction: standard setup

- *reference* semantics:
  - must be natural
  - easy to define
  - non-compositional

$\Rightarrow$ contextual, observational

- context $\mathcal{C}[\_]$= "program with a hole"
- filling the hole with a part of a program (component C): complete program $\mathcal{C}[C]$
- what is a context/component?: depends on the language/syntax (sequential/parallel/functional ... contexts)

F-A: standard setup (cont'd)

- given a closed program P: $\mathcal{O}(P)$ = observations

$\Rightarrow$ observational equivalence:

$$\boxed{C_1 \equiv_{obs} C_2 \quad \text{iff} \quad \forall \mathcal{C}.\ \mathcal{O}(\mathcal{C}[C_1]) = \mathcal{O}(\mathcal{C}[C_2])}$$

- given a denotational semantics $[\![\_]\!]_{\mathcal{D}}$, resp. the implied equality $\equiv_{\mathcal{D}}$

$\Rightarrow$ $\equiv_{\mathcal{D}}$ is fully abstract wrt. $\equiv_{obs}$:

$$\boxed{\equiv_{obs} \quad = \quad \equiv_{\mathcal{D}}}$$

Introduction

Calculus

Classes and observable behavior

Object connectivity

Conclusion

Object calculus: informal

- formal model(s) of oo languages
- in the tradition of the λ-calculi, process calculi . . .
- more specifically:
  - object-calculi of Abadi/Cardelli [1]
  - π-calculus: processes, parallelism, name-passing [5][8]
  - ν-calculus: λ-calc. with name creation (references) respectively its concurrent version [6][2]

Concurrent ν-calculus with classes

- program = "set" of named threads, objects, and classes: $n\langle t\rangle$, $n[c]$ and $n[\!(l_1 = m_1, \ldots, l_k = m_k)\!]$
- dynamic scoping of names
  - $\nu n{:}T.\ (C_1 \parallel C_2)$
  - communication of names changes the scope ("scope extrusion")
- class = "like" an object that accepts only a *new*-method; class names are not first-class citizens, i.e. not subject to
  - storing, sending, receiving etc.
- methods = functions with specific "self"-parameter[1]
- active entities: threads
  - sequencing + local, static scoping: $let\ x = e\ in\ t$
  - thread creation

[1] In the presence of subtyping, the parameter would be late-bound.

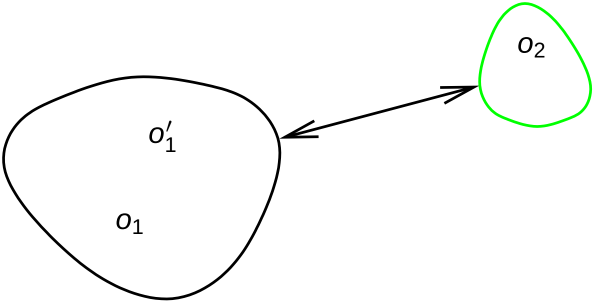
Concurrent ν-calculus with classes

$$
\begin{array}{rcll}
C & ::= & \mathbf{0} \mid C \parallel C \mid \nu(n{:}T).C \mid n[\![O]\!] \mid n[n,F] \mid n\langle t\rangle & \text{program} \\
O & ::= & M, F & \text{object} \\
M & ::= & l = m, \ldots, l = m & \text{method suite} \\
F & ::= & l = f, \ldots, l = f & \text{fields} \\
m & ::= & \varsigma(n{:}T).\lambda(x{:}T, \ldots, x{:}T).t & \text{method} \\
f & ::= & \varsigma(n{:}T).\lambda().v & \text{field} \\
t & ::= & v \mid \mathit{stop} \mid \mathit{let}\, x{:}T = e\, \mathit{in}\, t & \text{thread} \\
e & ::= & t \mid \text{if } v = v \text{ then } e \text{ else } e & \text{expr.} \\
 & \mid & v.l(v, \ldots, v) \mid n.l := v \mid \mathit{currentthread} & \\
 & \mid & \mathit{new}\, n \mid \mathit{new}\langle t\rangle & \\
v & ::= & x \mid n & \text{values}
\end{array}
$$

Semantics

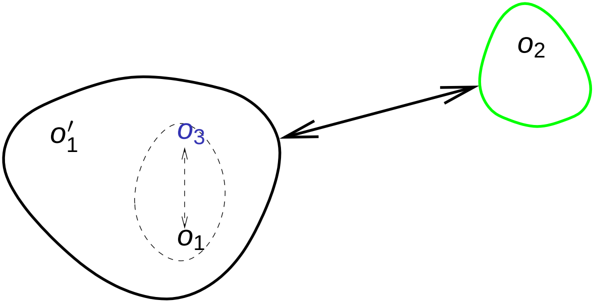
- given in various "stages"
  - internal (component-local) steps
  - external, global steps, interacting with the environment
  - computation steps modulo α-conversion
- typed operational semantics

Internal steps



- black: objects of the component
- green: objects of the environment

Internal steps



- o_1 creates an internal object o_3 (assume: thread n visits o_1)

$$
\begin{array}{l}
\mathbf{c}[\![F, M]\!] \parallel n\langle let\, x{:}c = new\, \mathbf{c}\, in\, t\rangle \rightsquigarrow \\
c[\![F, M]\!] \parallel \nu \mathbf{o_3}{:}\mathbf{c}.\ (n\langle let\, x{:}c = \mathbf{o_3}\, in\, t\rangle \parallel \mathbf{o_3}[\mathbf{c}, \mathbf{F}]) \ldots
\end{array}
$$

Semantics: Internal steps

- 3 exemplary axioms
- confluent ($\rightsquigarrow$) and non-confluent ($\xrightarrow{\tau}$) internal steps
- for $\textsc{Call}_i$: $M.l(o)(\vec{v})\,in\,t$: parameter passing, especially replacing the ς-bound self-parameter by o.

$$n\langle let\ x{:}T = \mathbf{v}\ in\ t\rangle \rightsquigarrow n\langle t[\mathbf{v}/\mathbf{x}]\rangle$$

$$\mathbf{c}[\![F, M]\!] \parallel n\langle let\ x{:}c = new\ \mathbf{c}\ in\ t\rangle \rightsquigarrow c[\![F, M]\!] \parallel \nu\mathbf{o}{:}\mathbf{c}.\ (n\langle let\ x{:}c = \mathbf{o}\ in\ t\rangle \parallel \mathbf{o}[\mathbf{c}, \mathbf{F}]\ )$$

$$c[\![F, M]\!] \parallel o[c, F'] \parallel n\langle let\ x{:}T = o.l(\vec{v})\ in\ t\rangle \xrightarrow{\tau} c[\![F, M]\!] \parallel o[c, F'] \parallel n\langle let\ x{:}T = M.l(o)(\vec{v})\ in\ t\rangle$$

Semantics: External steps

- "typed" operational semantics
- i.e., labeled steps between typing judgments: $\Delta \vdash P : \Theta$
  - Δ = "assumptions"
    - names assumed present in the rest
  - Θ = "commitments"
    - names guaranteed to the rest
- steps labeled by
  - thread id
  - communicated values
  - kind of communication (!/?, call/return)

External steps (2)

- e.g.: outgoing calls and incoming returns

$$\frac{o \in \Delta}{\Delta \vdash C \parallel n\langle let\, x{:}T = o.l(\vec{v})\, in\, t\rangle : \Theta \xrightarrow{n\langle call\ o.l(\vec{v})\rangle!} \Delta \vdash C \parallel n\langle let\, x{:}T = block\, in\, t\rangle : \Theta}$$

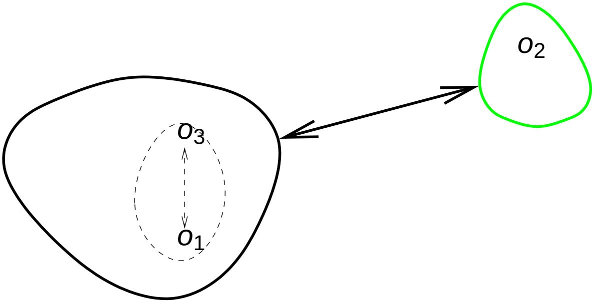
$$\frac{;\Delta, \Theta \vdash v : T}{\Delta \vdash C \parallel n\langle let\, x{:}T = block\, in\, t\rangle : \Theta \xrightarrow{n\langle return(v)\rangle?} \Delta \vdash C \parallel n\langle t[v/x]\rangle : \Theta}$$

External steps: Scoping

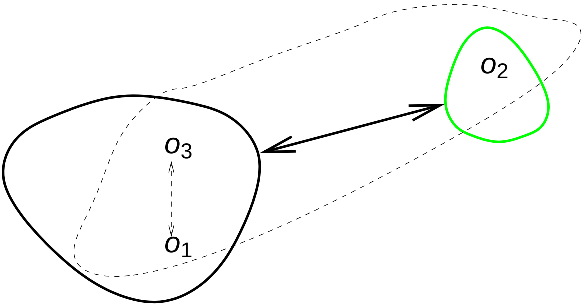
- names
  - for object and thread id's
  - can be generated freshly: "*new*"
  - valid within dynamic scopes
  - up-to renaming
- dynamic, i.e.,
  - names can be sent around: scope is extended
  - also: across component interface
  - bound exchange of names: "ν"

External steps: Scoping

internal o_3 is sent outside, e.g., as argument of method call to external o_2



External steps: Scoping



$$\Delta \vdash \nu o_3{:}c_3.(n\langle o_2.l(o_3); t\rangle \parallel o_1[\ldots]) : \Theta \xrightarrow{\nu o_3{:}c_3.\ n\langle call\ \ o_2.l(o_3)\rangle!}$$
$$\Delta \vdash n\langle block; t\rangle \parallel o_3[\ldots] : \Theta, o_3{:}c_3$$

F-A in an object-based conc. setting

- [3]: for the concurrent ν-calculus
- notion of observation: may-testing equivalence.[1]
  Formalized here: whether a specific context method ("$o.success()$") is called
- component = set of parallelly "running" objects + threads
- observable: message exchange at the boundary

$\Rightarrow$ fully abstract observable behavior = communication traces of the labels of the OS

[1] actually: they use may-preorder.

What changes?

- classes are units of exchange: $\mathcal{C}[n[\![O]\!]]$!
- i.e., internal and external classes
- component objects can instantiate external classes

can one use these objects for “observations”?

- instances of external classes,
  - instantiation itself is unobservable
  - comm. between component and object observable
  - but:
    - their existence is (principally) unknown to the rest of environment ($\neq$ OC),
    - unless the component gives away their identity!

Completeness: line of argument

- goal: if $C_1 \equiv_{obs} C_2$, then $C_1 \equiv_{\mathcal{D}} C_2$
- so, given a legal trace $s \in [\![C_1]\!]_{\mathcal{D}}$, do
  - construct a complementary context $\mathcal{C}_{\bar{s}}$
  - composition: program + context do the observation

$$\mathcal{C}_{\bar{s}}[C_1] \longrightarrow^* success$$

  - observational equivalence: C_2 can do that, too:

$$\mathcal{C}_{\bar{s}}[C_2] \longrightarrow^* success$$

  - decomposition:[2] $s \in [\![C_2]\!]_{\mathcal{D}}$

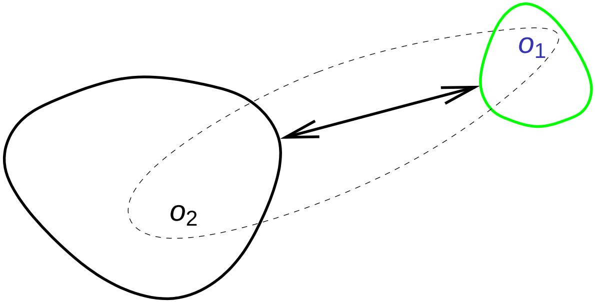
$\Rightarrow$ problems for completeness (apart from technicalities)

1. definability $\Rightarrow$ what are legal traces?
2. what can be observed/distinguished?

[2] That s is a trace of C_2 by decomposition is not a direct consequence. I ignore that here.

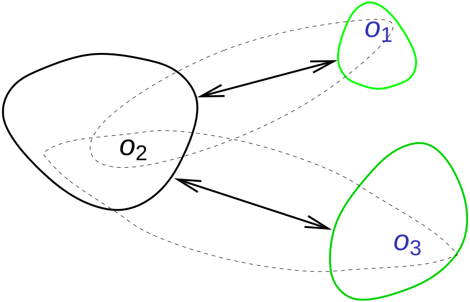
Impossible incoming names?

- Assume: component instantiates two external classes (into o_1 and o_3)



- can o_1 and o_3 be sent in the same argument list? (for example)

Impossible incoming names?



- trace labelled

$$\nu o_1.creates\ o_1!.\ \nu o_3.creates\ o_3!.\ n'\langle call\ o_2.l(o_1, o_3)\rangle?$$

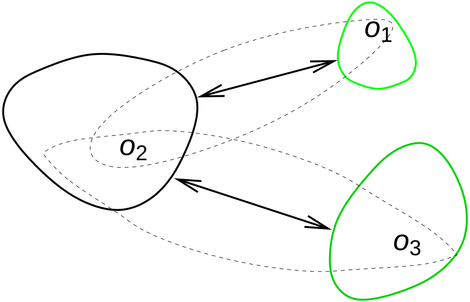
impossible!

Acquaintance

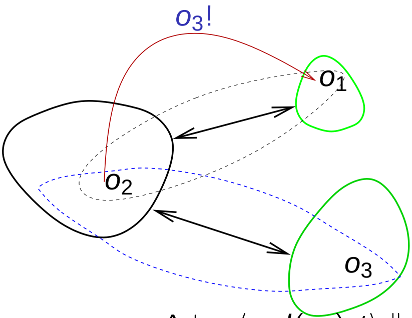
- o_1 and o_3
  - cannot occur in the same label and
  - cannot determine the order of events mutually,

  because they don't "know" of each other
- if "connected", they
  - could occur in the same label and
  - could (in principle) cooperate to observe order of interaction
- connectivity or "acquaintance" is dynamic
- the only one to make o_1 and o_3 acquainted: the component

Dynamic acquaintance



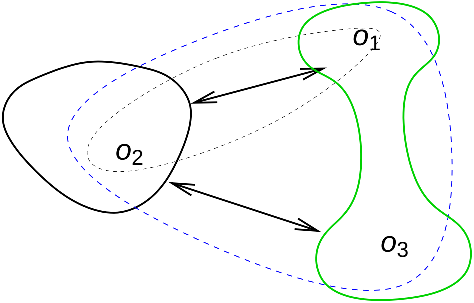
Dynamic acquaintance



$$\Delta \vdash n\langle o_1.l(o_3); t\rangle \parallel o_2[\ldots] : \Theta \xrightarrow{n\langle call\ \ o_1.l(o_3)\rangle!} \Delta \vdash n\langle block; t\rangle \parallel o_2[\ldots] : \Theta$$

- no scope extrusion from the (global) perspective of the component

Dynamic acquaintance



- scope enlarged
- o_1 knows o_3

$\Rightarrow$ o_3 could know now o_1, too

- and all objects that o_3 knows, could know o_1 in turn, too ...

Acquaintance: assumptions and commitments

- acquaintance = equivalence relation on object id's

$\Rightarrow$ keep track of (the worst-case) of connectivity

$\Rightarrow$ set of "equations"; clique: implied equational theory

- e.g., sending o_1 to o_2, adds $o_1 \hookrightarrow o_2$ to the equations

Approximating the mutual knowledge

- component keeps book about “whom it told what”
- transitions

$$\Delta; E_\Delta \vdash C : \Theta; E_\Theta \xrightarrow{a} \acute{\Delta}; \acute{E}_\Delta \vdash \acute{C} : \acute{\Theta}; \acute{E}_\Theta$$

- Assumption context: $E_\Delta \subseteq \Delta \times (\Delta + \Theta)$ = pairs of objects
- written $o_1 \hookrightarrow o_2$:
- worst case: equational theory implied by E_Δ (on Δ):

$$E_\Delta \vdash o_1 \leftrightharpoons o_2$$

(for $o_2 \in \Theta$: $E_\Delta \vdash o_1 \leftrightharpoons ; \hookrightarrow o_2$)

Scoping & lazy instantiation

- object names get known “on the other side” $\Rightarrow$ scope extrusion
- external classes $\Rightarrow$ cross border instantiation
  - instance created “on the other side”
  - reference kept “at this side”
- instantiation itself: not observable $\Rightarrow$ lazy instantiation

$$\Delta \qquad \vdash \nu(o_2{:}c_2)(n\langle o_1.l(o_2); t\rangle \parallel o_2[\ldots] \parallel \ldots) : \Theta, c_2{:}T_2$$

- label $a = \nu(\Theta', \Delta').\ n\langle call\ o_2.l(\vec{v})\rangle!$

Scoping & lazy instantiation

- object names get known "on the other side" $\Rightarrow$ scope extrusion
- external classes $\Rightarrow$ cross border instantiation
  - instance created "on the other side"
  - reference kept "at this side"
- instantiation itself: not observable $\Rightarrow$ lazy instantiation

$$\Delta \vdash \nu(o_2{:}c_2)(n\langle o_1.l(o_2); t\rangle \parallel o_2[\ldots] \parallel \ldots) : \Theta, c_2{:}T_2$$

$$\xrightarrow{\nu(o_2:c_2).n\langle call\ o_1.l(o_2)\rangle!}$$

- label $a = \nu(\Theta', \Delta').\ n\langle call\ o_2.l(\vec{v})\rangle!$

Scoping & lazy instantiation

- object names get known “on the other side” ⇒ scope extrusion
- external classes ⇒ cross border instantiation
  - instance created “on the other side”
  - reference kept “at this side”
- instantiation itself: not observable ⇒ lazy instantiation

$$\Delta \quad \vdash \nu(o_2{:}c_2)(n\langle o_1.l(o_2); t\rangle \parallel o_2[\ldots] \parallel \ldots) : \Theta, c_2{:}T_2$$

$$\xrightarrow{\nu(o_2:c_2).n\langle call\ o_1.l(o_2)\rangle!}$$

$$\Delta \quad \vdash n\langle block; t\rangle \parallel o_2[\ldots] \parallel \ldots : \Theta, c_2{:}T_2, o_2{:}c_2$$

- label $a = \nu(\Theta', \Delta').\ n\langle call\ o_2.l(\vec{v})\rangle!$

Scoping & lazy instantiation

- object names get known "on the other side" $\Rightarrow$ scope extrusion
- external classes $\Rightarrow$ cross border instantiation
  - instance created "on the other side"
  - reference kept "at this side"
- instantiation itself: not observable $\Rightarrow$ lazy instantiation

$$\Delta, c_2{:}T_2 \vdash \nu(o_2{:}c_2)(n\langle o_1.l(o_2); t\rangle \qquad\qquad \parallel \ldots) : \Theta$$

- label $a = \nu(\Theta', \Delta').\ n\langle call\ o_2.l(\vec{v})\rangle!$

Scoping & lazy instantiation

- object names get known “on the other side” $\Rightarrow$ scope extrusion
- external classes $\Rightarrow$ cross border instantiation
  - instance created “on the other side”
  - reference kept “at this side”
- instantiation itself: not observable $\Rightarrow$ lazy instantiation

$$\Delta, c_2{:}T_2 \vdash \nu(o_2{:}c_2)(n\langle o_1.l(o_2); t\rangle \qquad \parallel \ldots) : \Theta$$

$$\xrightarrow{\nu(o_2:c_2).n\langle call\ o_1.l(o_2)\rangle!}$$

- label $a = \nu(\Theta', \Delta'). \, n\langle call\ o_2.l(\vec{v})\rangle!$

Scoping & lazy instantiation

- object names get known "on the other side" $\Rightarrow$ scope extrusion
- external classes $\Rightarrow$ cross border instantiation
  - instance created "on the other side"
  - reference kept "at this side"
- instantiation itself: not observable $\Rightarrow$ lazy instantiation

$$\Delta, c_2{:}T_2 \vdash \nu(o_2{:}c_2)(n\langle o_1.l(o_2); t\rangle \qquad\qquad \parallel \ldots) : \Theta$$

$$\xrightarrow{\nu(o_2:c_2).n\langle call\ o_1.l(o_2)\rangle!}$$

$$\Delta, c_2{:}T_2, o_2{:}T_2 \vdash n\langle block; t\rangle \parallel \qquad\qquad \parallel \ldots : \Theta$$

- label $a = \nu(\Theta', \Delta').\ n\langle call\ o_2.l(\vec{v})\rangle!$

Legal traces

- core of completeness: definability $\Rightarrow$
- for each legal trace s: construct a component C_s realizing it
- thus first: characterize the legal traces
- derivability of legal-trace-judgement:

$$\boxed{\Delta; E_\Delta \vdash r \rhd s : \mathit{trace}\, \Theta; E_\Theta}$$

Legal traces: incoming call

- General setup: scan the trace, where
  - r: history
  - as future with next label a

$$\frac{\text{lots of conditions} \qquad \acute{\Delta}; \acute{E}_{\Delta} \vdash \quad r\ a \rhd s \quad : \mathit{trace}\ \acute{\Theta}; \acute{E}_{\Theta}}{\Delta; E_{\Delta} \vdash \quad r \rhd a\ s \quad : \mathit{trace}\ \Theta; E_{\Theta}}$$

"Lots of conditions"

- For completeness: component must realize all possible traces but not more!
- various aspects
  - "global": call-return discipline = balanced/"parenthetic" (per thread)
  - "local"
    - no name clashes: scoping/renaming
    - well-typedness
    - impossible name communication ("connectivity")

Incoming call: acquaintance

- let $a = n\langle call\ o_2.l(\vec{v})\rangle?$

$$\frac{\acute{E}_{\Theta} = E_{\Theta} + o_2 \hookrightarrow \vec{v} \\ \Delta; E_{\Delta} \vdash r\ a \quad \rhd s : trace\,\Theta; \acute{E}_{\Theta} \qquad E_{\Delta} \vdash v_i \leftrightharpoons v_j}{\Delta; E_{\Delta} \vdash r \rhd a\, s : trace\,\Theta}$$

- caller anonymous $\Rightarrow$ not mentioned in label
- nonetheless: needed for bookkeeping: to return to the same caller

$\Rightarrow$ remembered in the history

Incoming call: Who's the caller?

- let $\quad a = n\langle call\ o_2.l(\vec{v})\rangle?$

$$\frac{\acute{E}_\Theta = E_\Theta + o_2 \hookrightarrow \vec{v} \qquad \Delta; E_\Delta \vdash r\ a \quad \rhd s : trace\,\Theta; \acute{E}_\Theta \qquad E_\Delta \vdash v_i \leftrightharpoons v_j}{\Delta; E_\Delta \vdash r \rhd a\ s : trace\,\Theta}$$

- caller anonymous $\Rightarrow$ not mentioned in label
- nonetheless: needed for bookkeeping: to return to the same caller

$\Rightarrow$ remembered in the history

Incoming call: Who's the caller?

- let $\quad a = n\langle call\ o_2.l(\vec{v})\rangle?$

$$\frac{\acute{E}_\Theta = E_\Theta + o_2 \hookrightarrow \vec{v} \qquad \Delta; E_\Delta \vdash r\ a_{o_1} \rhd s : trace\ \Theta; \acute{E}_\Theta \qquad \Delta; E_\Delta \vdash o_1 \leftrightharpoons\hookrightarrow \vec{v}, o_2 : \Theta; E_\Theta}{\Delta; E_\Delta \vdash r \rhd a\ s : trace\ \Theta}$$

- caller anonymous $\Rightarrow$ not mentioned in label
- nonetheless: needed for bookkeeping: to return to the same caller

$\Rightarrow$ remembered in the history

Incoming call: scoping

- object names get known "on the other side" $\Rightarrow$ scope extrusion
- external classes $\Rightarrow$ cross border instantiation
  - instance created "on the other side"
  - reference kept "at this side"
- instantiation itself: not observable $\Rightarrow$ lazy instantiation
- label $a = \nu(\Delta', \Theta').\, n\langle call\ o_2.l(\vec{v})\rangle?$

$$\frac{\begin{array}{c}\Delta \vdash o_1 : c_1 \\ \acute{\Theta}; \acute{E}_{\Theta} = \Theta; E_{\Theta} + (\Theta'; o_2 \hookrightarrow \vec{v}) \qquad \acute{\Delta}; \acute{E}_{\Delta} = \Delta; E_{\Delta} + \Delta'; o_1 \hookrightarrow (\Delta', \Theta') \\ dom(\Delta', \Theta') \subseteq fn(n\langle call\ o_2.l(\vec{v})\rangle) \\ \acute{\Delta}; \acute{E}_{\Delta} \vdash o_1 \rightleftharpoons\!\hookrightarrow \vec{v}, o_2 : \acute{\Theta} \qquad \acute{\Delta}; \acute{E}_{\Delta} \vdash r\ a_{o_1} \rhd s : trace\ \acute{\Theta}; \acute{E}_{\Theta}\end{array}}{\Delta; E_{\Delta} \vdash r \rhd a\, s : trace\ \Theta; E_{\Theta}}$$

Legal traces: balance

- incoming call
- check for input enabledness per thread
- consult the history
- for instance: incoming return *a* possible in a next step

$$\frac{pop\ n\ r = \nu(\Theta').\ n\langle call\ o_2.l(\vec{v})\rangle!}{\Delta \vdash r \rhd \nu(\Delta').\ n\langle return(v)\rangle? : \Theta}$$

- before a return: there must have been an outgoing call
- *pop* picks out the last "matching" call

Incoming comm.: the full story

$$
\frac{\begin{array}{c}
a = \nu(\Delta', \Theta').\, n\langle \mathit{call}\ o_2.l(\vec{v})\rangle? \qquad \Delta \vdash o_1 : c_1 \qquad \Delta \vdash r \rhd a : \Theta \\
\acute{\Theta}; \acute{E}_\Theta = \Theta; E_\Theta + (\Theta'; o_2 \hookrightarrow \vec{v}, n \hookrightarrow o_2) \quad \acute{\Delta}; \acute{E}_\Delta = \Delta; E_\Delta + \Delta'; o_1 \hookrightarrow (\Delta', \Theta') \setminus n \\
; \acute{\Theta} \vdash o_2 :: [\![(\ldots, l{:}\vec{T} \to T, \ldots)]\!] \quad \acute{\Delta} \vdash n{:}\, \mathit{thread} \quad \acute{\Theta} \vdash n{:}\, \mathit{thread} \quad ; \acute{\Delta}, \acute{\Theta} \vdash \vec{v} : \vec{T} \\
\mathit{dom}(\Delta', \Theta') \subseteq \mathit{fn}(n\langle \mathit{call}\ o_2.l(\vec{v})\rangle) \\
\acute{\Delta}; \acute{E}_\Delta \vdash n \leftrightharpoons o_1 \leftrightharpoons\!\hookrightarrow \vec{v}, o_2 : \acute{\Theta} \qquad \acute{\Delta}; \acute{E}_\Delta \vdash r\ a_{o_1} \rhd s : \mathit{trace}\, \acute{\Theta}; \acute{E}_\Theta
\end{array}}{\Delta; E_\Delta \vdash r \rhd a\, s : \mathit{trace}\, \Theta; E_\Theta}
$$

Definability

- given a legal trace s $\Rightarrow$ define C_s by

  induction on the derivation for
  $$\Delta; E_\Delta \vdash r \rhd s : trace\, \Theta; E_\Theta$$

$\Rightarrow$ construct the program backwards!

actions on the commitment context E_Θ:

  E_Θ: each object knows its clique, kept up-to date
  - giving away new id's: create them
    propagate/broadcast information through the clique

- incoming calls: wrap up the method body, put it into the class

Definability/synchronization

- for example outgoing call $a = \nu(\Theta', \Delta').\, n\langle call\ o_2.l(\vec{v})\rangle!$
- we know: afterwards

$$\begin{aligned} \acute{C}_s &= n\langle o_1 \text{ } blocks\ for\ o_2; t'\rangle \parallel C'_s \\ \acute{E}_\Theta &= E_\Theta + o_1 \hookrightarrow \Theta' \end{aligned}$$

- construct component $\grave{C}_s$ before the call:

$$\grave{C}_s = C'_s \parallel n\langle t^o_{sync}(\Theta', \check{a}); o_2.l(\vec{v})\rangle$$

where

$$\begin{aligned} t^o_{sync}(\Theta', \check{a}) \triangleq (\!|\ let\ \ & \vec{x}{:}\vec{c} = new(\Theta') \\ in\ \ & pick(\vec{x}); \\ & self.propagate(\vec{x}); \\ & \check{a} \triangleright |\!)\ . \end{aligned}$$

What can be observed, then?

- observers: not just one static outside context but
- dynamic cliques of acquainted objects
  - existing cliques only grow larger: merging
  - new ones can be created by the component
- for full-abstraction:
  - traces per clique (in first approximation)
  - worst-case: “conspiracy” of environment
- acquaintance = equivalence relation on object id’s

$\Rightarrow$ component keeps track of (the worst-case) of cliques $\Rightarrow$ set of equations; clique: implied equational theory

- e.g., sending o_1 to o_2, adds $o_1 \hookrightarrow o_2$ to the equations

Introduction

Calculus

Classes and observable behavior

Object connectivity

Conclusion

Conclusions

- are classes good composition units?
- what about cloning?
- lock-synchronization
- subtype polymorphism & subclassing

References I

[1] M. Abadi and L. Cardelli.
A Theory of Objects.
Monographs in Computer Science. Springer, 1996.

[2] A. D. Gordon and P. D. Hankin.
A concurrent object calculus: Reduction and typing.
In U. Nestmann and B. C. Pierce, editors, *Proceedings of HLCL '98*, volume 16.3 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers, 1998.

[3] A. Jeffrey and J. Rathke.
A fully abstract may testing semantics for concurrent objects.
In *Proceedings of LICS '02*. IEEE, Computer Society Press, July 2002.

[4] R. Milner.
Fully abstract models of typed λ-calculi.
Theoretical Computer Science, 4:1–22, 1977.

[5] R. Milner, J. Parrow, and D. Walker.
A calculus of mobile processes, part I/II.
Information and Computation, 100:1–77, Sept. 1992.

[6] A. M. Pitts and D. B. Stark.
Observable properties of higher-order functions that dynamically create local names, or: What's new.
In A. M. Borzyszkowski and S. Sokołowski, editors, *Proceedings of MFCS '93*, volume 711 of *Lecture Notes in Computer Science*, pages 122–141. Springer-Verlag, Sept. 1993.

[7] G. Plotkin.
LCF considered as a programming language.
Theoretical Computer Science, 5:223–255, 1977.

References II

[8] D. Sangiorgi and D. Walker.
The π-calculus: a Theory of Mobile Processes.
Cambridge University Press, 2001.