First-class Implementations

Turtling down Runtime Meta-levels

and PCLSRing up

François-René Rideau, *TUNES Project*

BostonHaskell, 2016-05-18

http://fare.tunes.org/files/cs/fci-bh2016.pdf

This Talk

Salvaged from my aborted 1999 PhD thesis:

The Semantics of Reflective Systems

Cousot at ENS was teaching Abstract Interpretation.

All I was interested in was the opposite direction:

Concrete Implementation

The Take Home Points

Implementation is co-(Abstract Interpretation)

Safe points are a key concept

First-class: the opposite of magic

First-class safe points (= PCLSRing!)

Applications: Migration, Optimistic Evaluation, etc.

Composing implementations for fun and profit

Runtime meta-programming brings new modularity

Plan

Implementation: Formalizing the notion

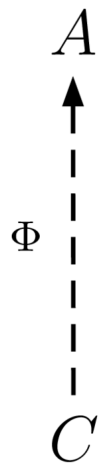
First-class Implementation protocol

Applications of First-class Implementations

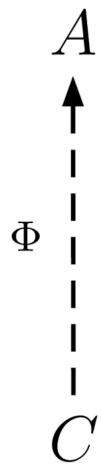
Runtime meta-programming architecture

I. Formalizing the notion of Implementation

Abstract Interpretation



Concrete Implementation



Concrete Implementation vs Abstract Interpretation

Dynamic (Runtime) vs Static (Compile-time)

Operational Semantics vs Denotational Semantics

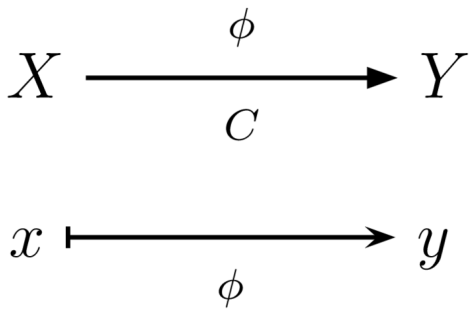
Downward (concrete) vs Upward (abstract)

Co-functorial vs Functorial

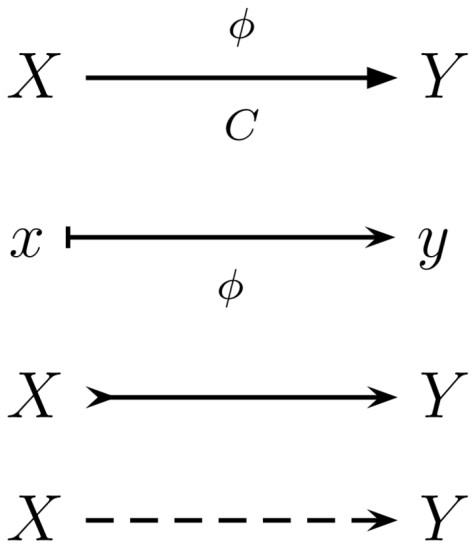
Noisy vs lossy

Non-deterministic vs deterministic

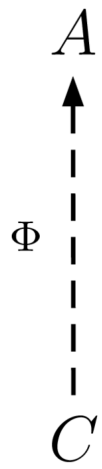
Categories



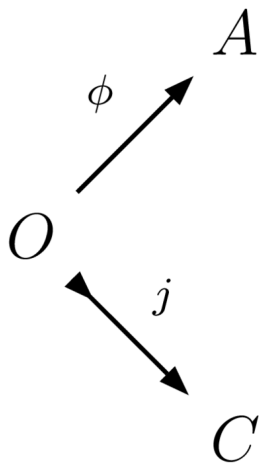
Categories



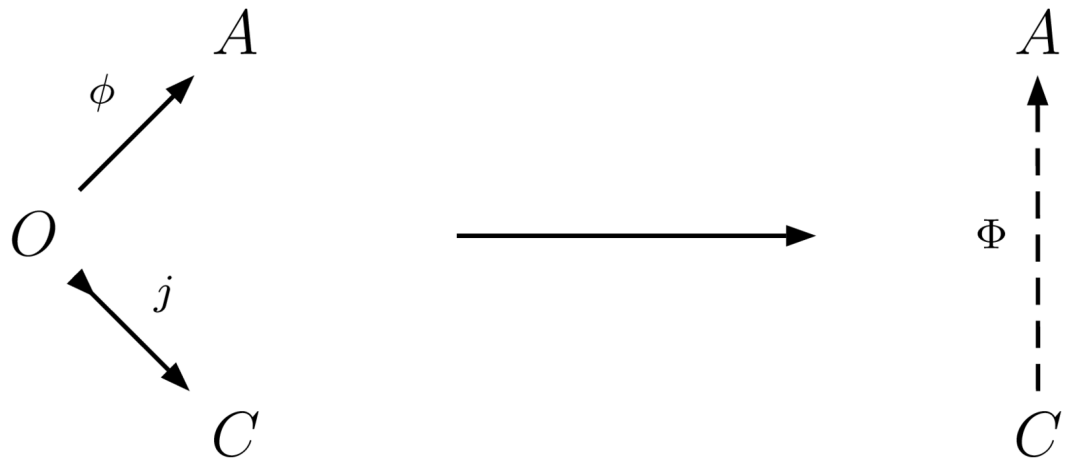
Partial Functions (1)



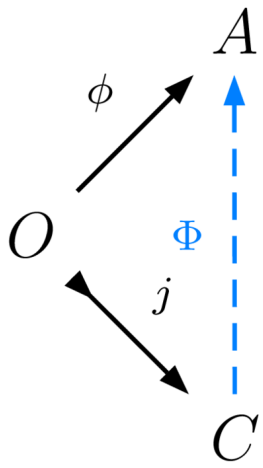
Partial Functions (2)



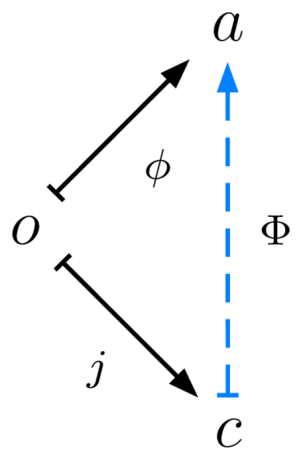
Partial Functions (3)



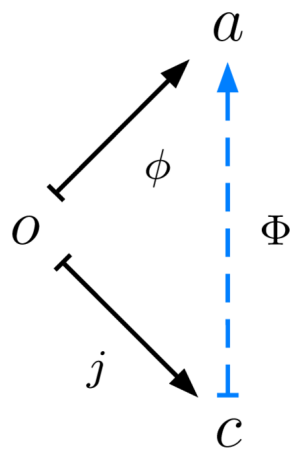
Deduction



Observable State

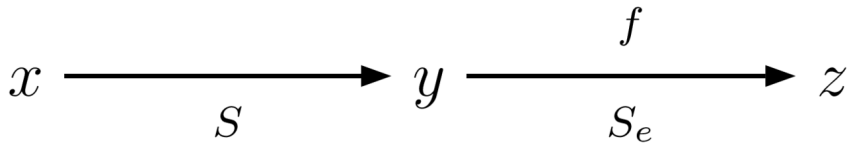


Observable State

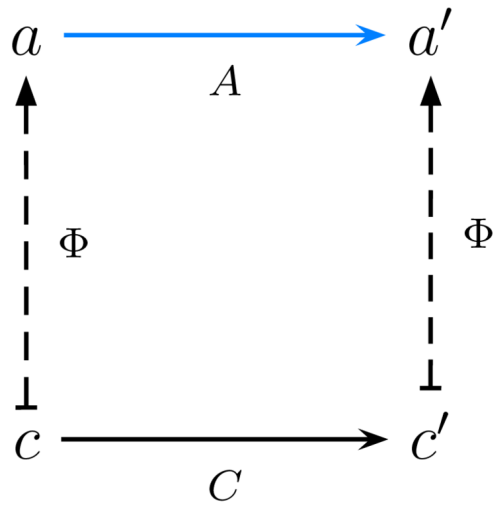


$o = c$

Operational Semantics

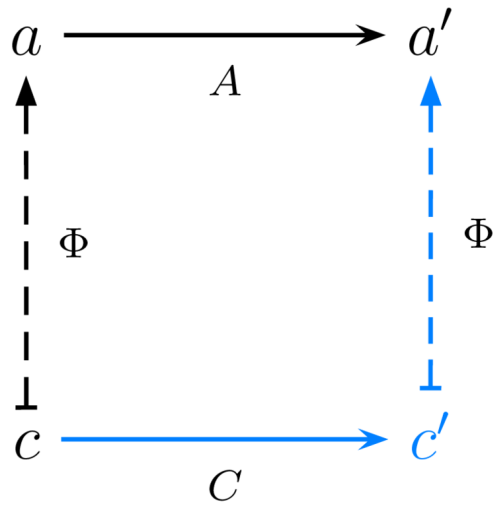


Soundness

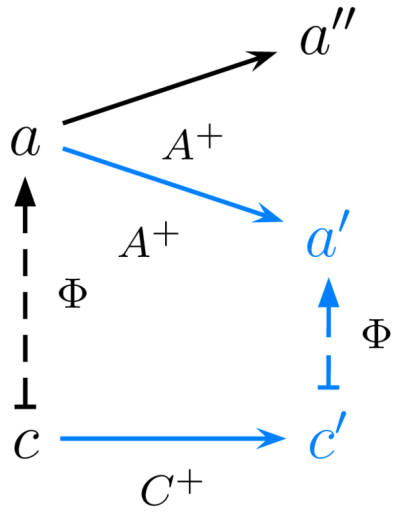


Totality

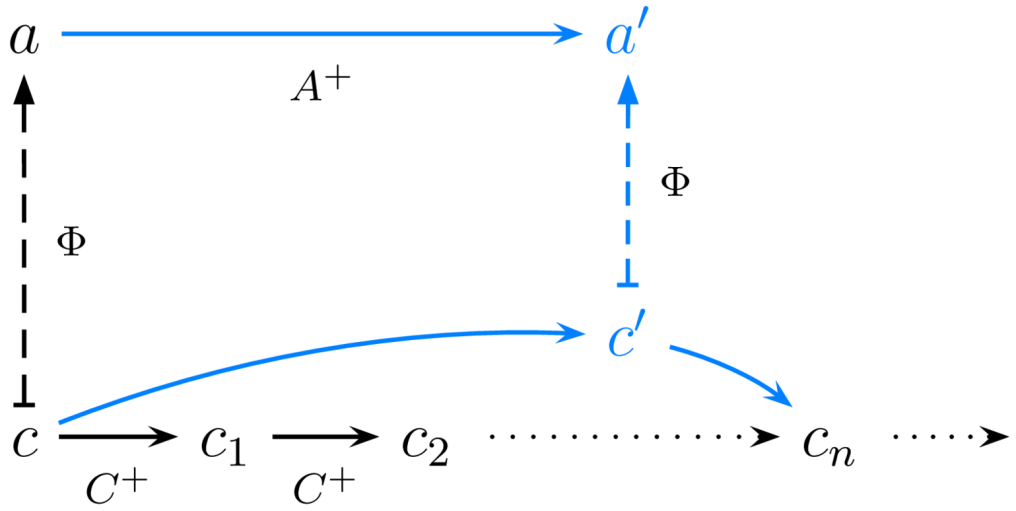
Completeness



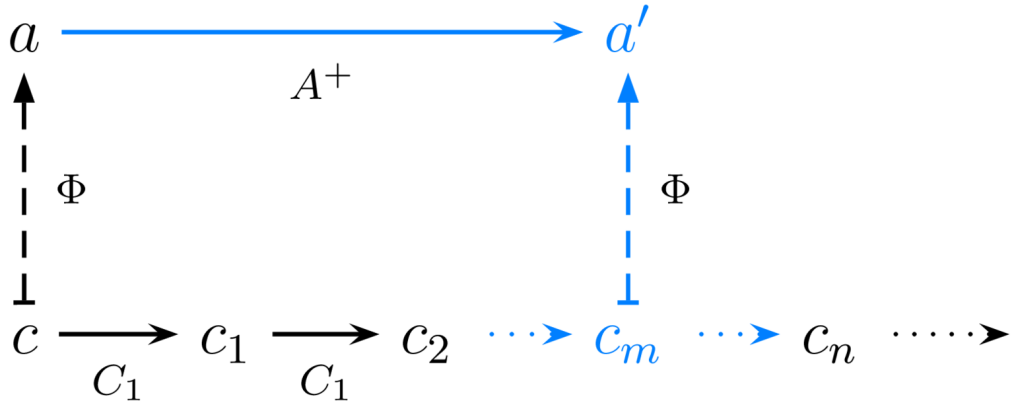
Advance Preservation



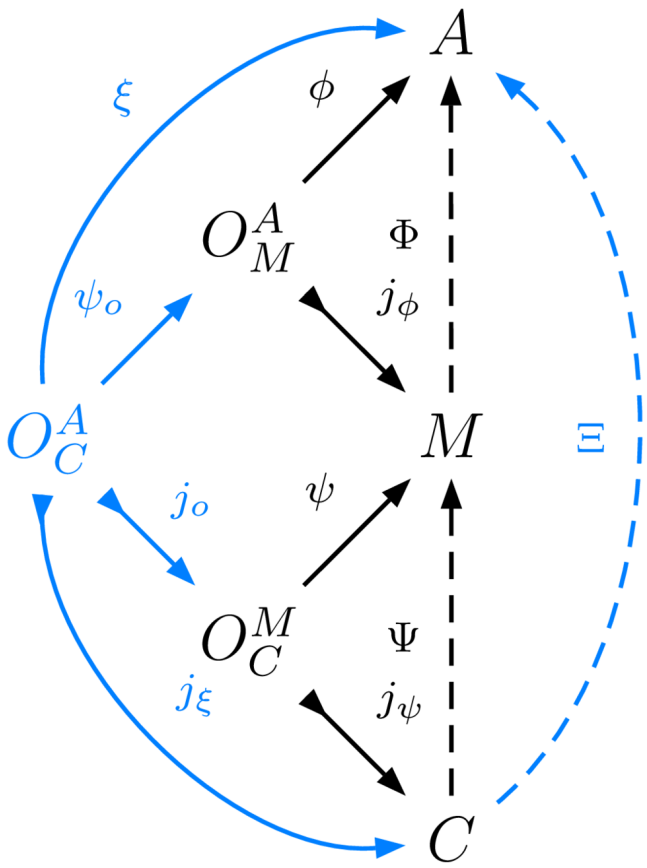
Liveness



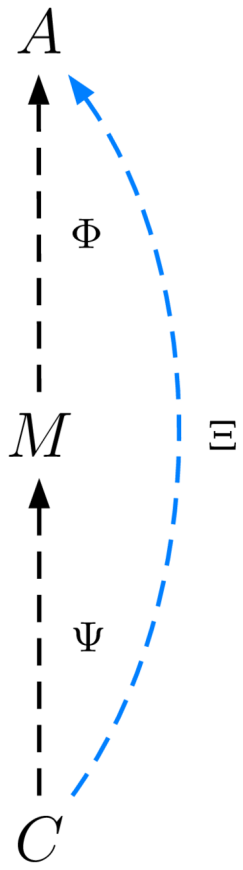
Strong Liveness



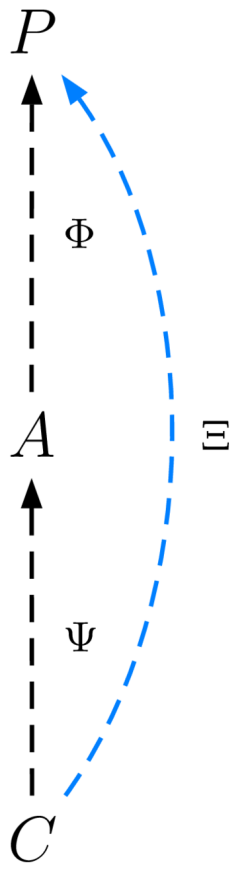
Composability



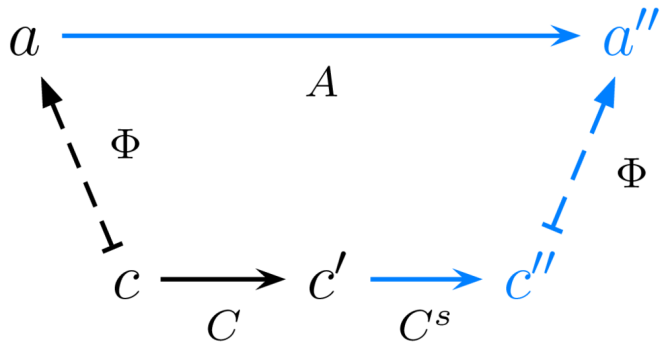
Composability



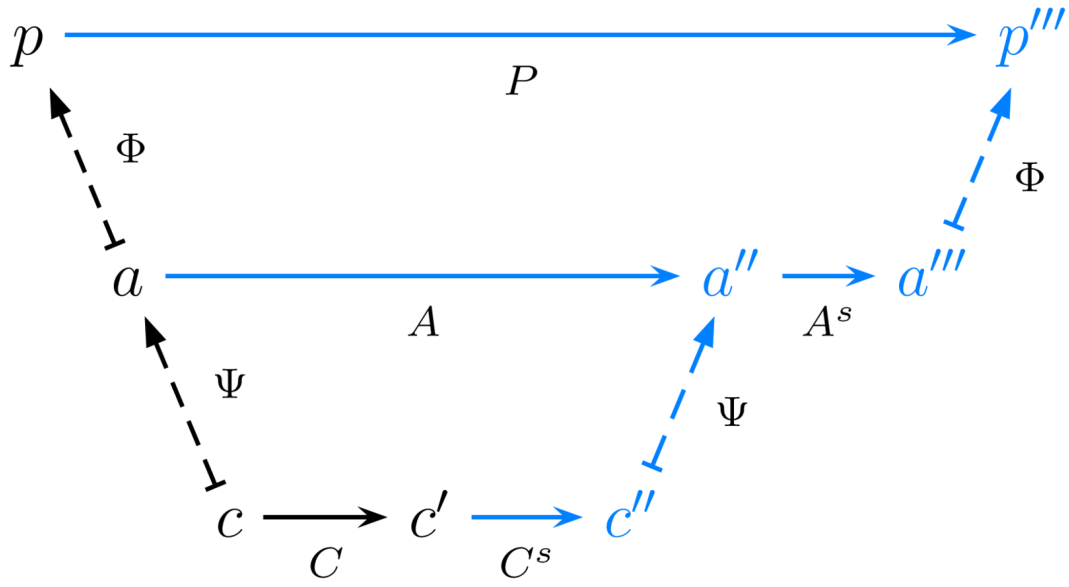
Composability



Observability (aka PCLSRing)



Observability (aka PCLSRing)



II. First-class Implementation protocol

Protocol: Categories

```
class Cat s where
  type Arr s :: *
  dom :: (Arr s) → s
  cod :: (Arr s) → s
  composeArr :: (Arr s) → (Arr s) → (Arr s)
  applyArr :: (Arr s) → s ⇢ s
```

Usual functions: →

Effectful functions: ⇢

Protocol: Operational Semantics

```
class (Cat s) ⇒ OpSem s where
  run :: s ⇢ Arr s
  done :: s → Bool
```

Protocol: Operational Semantics

```
class (Cat s) ⇒ OpSem s where
  run :: s ⇢ Arr s
  done :: s → Bool

  eval :: s ⇢ Arr s
  advance :: s ⇢ Arr s
```

Protocol: Implementation

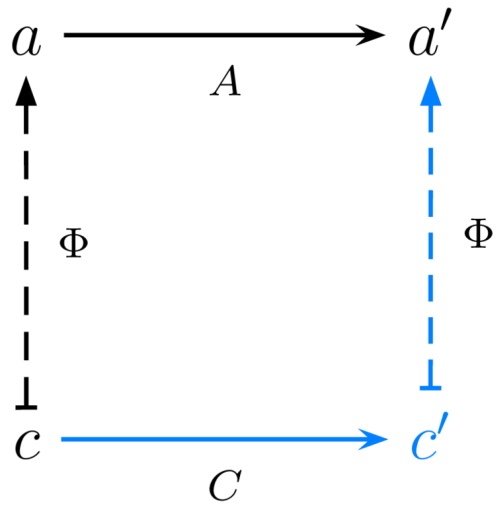
```
class Impl a c where
  interpret :: c ⇢ a
  mapInterpret :: (Arr c) ⇢ (Arr a)
```

So far, a functor from `c` to `a`

Protocol: Totality

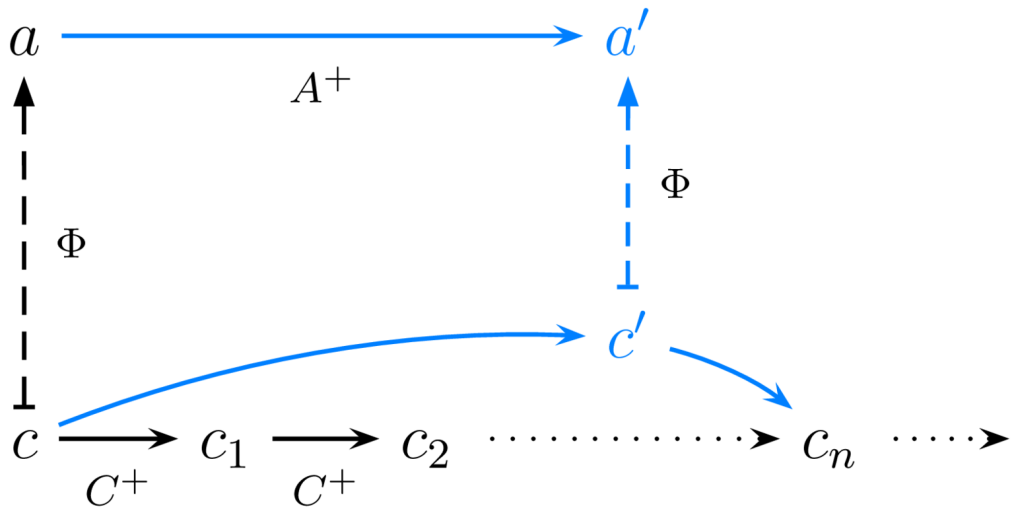

```
implement :: a ⇢ c
```

Protocol: Completeness



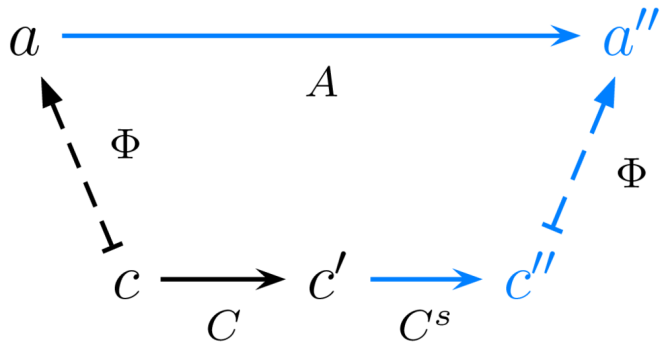

```
step :: c → (Arr a) ⇢ (Arr c)
```

Protocol: Liveness



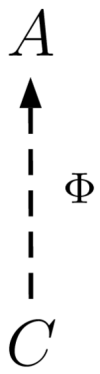

```
advanceInterpretation :: c ⇢ Arr c
```

Protocol: Observability (PCLSRing)



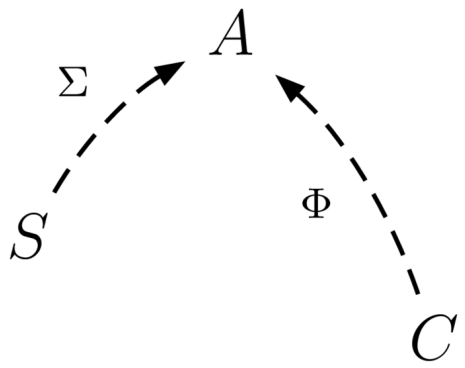

```
safePoint :: c ⇢ Arr c
```

Compilation (1)



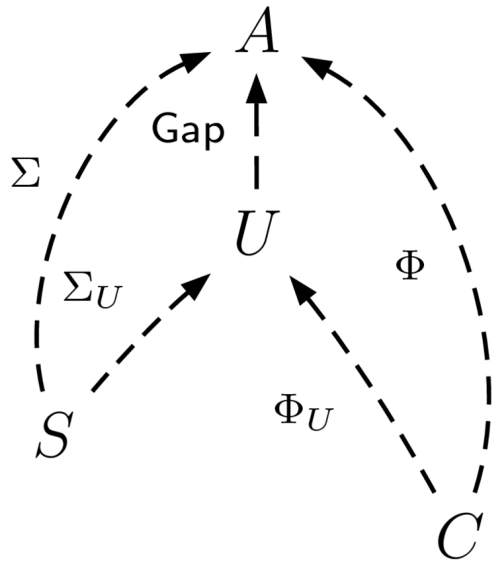

```
implement :: (Impl a c) ⇒ a ⇢ c
```

Compilation (2)



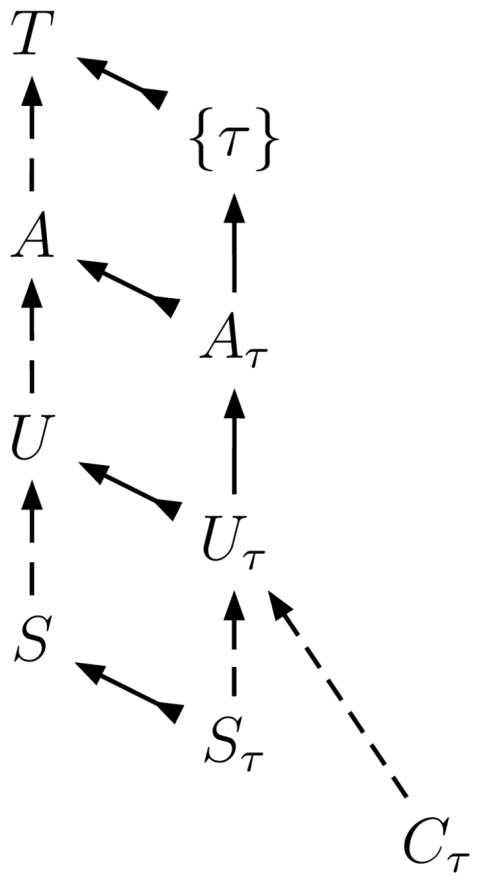

```
interpret :: (Impl a s) ⇒ s ⇢ a
implement :: (Impl a c) ⇒ a ⇢ c
```

Compilation (3)




```
u :: OpSem -- specify up to what rewrites
interpret :: (Impl u s) ⇒ s ⇢ u
implement :: (Impl u c) ⇒ u ⇢ c
```

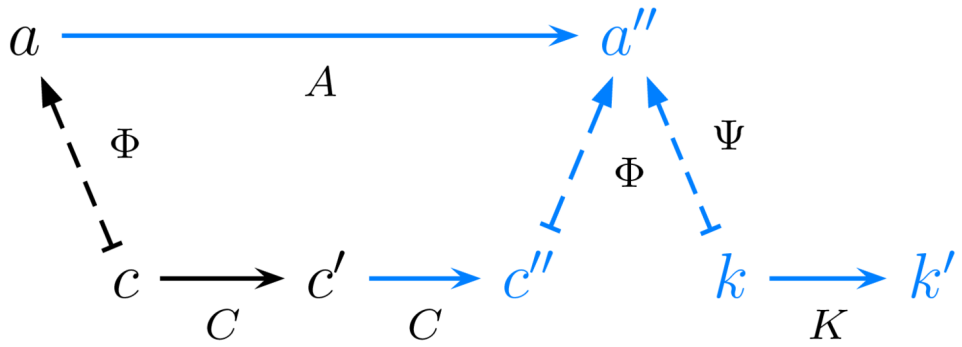
Static Type Systems



Subject reduction: *T* contains no exomorphisms

III. Applications of First-class Implementations

Migration



When your hammer is Migration...

Process Migration

Garbage Collection

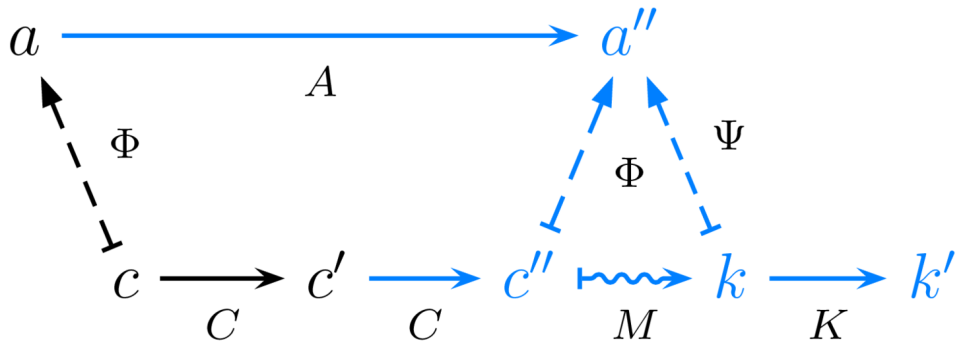
Zero Copy Routing

Dynamic Configuration

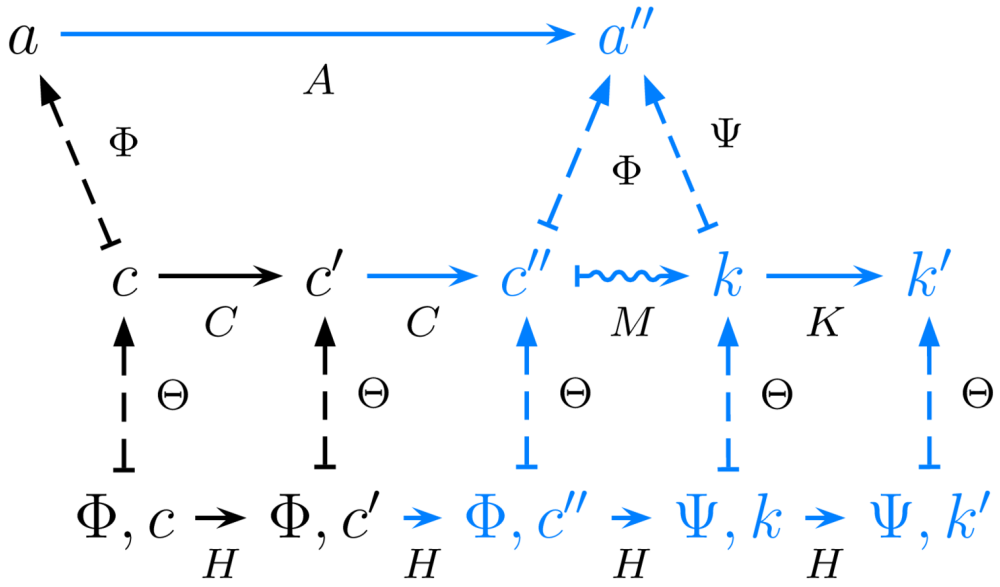
JIT Compilation

etc.

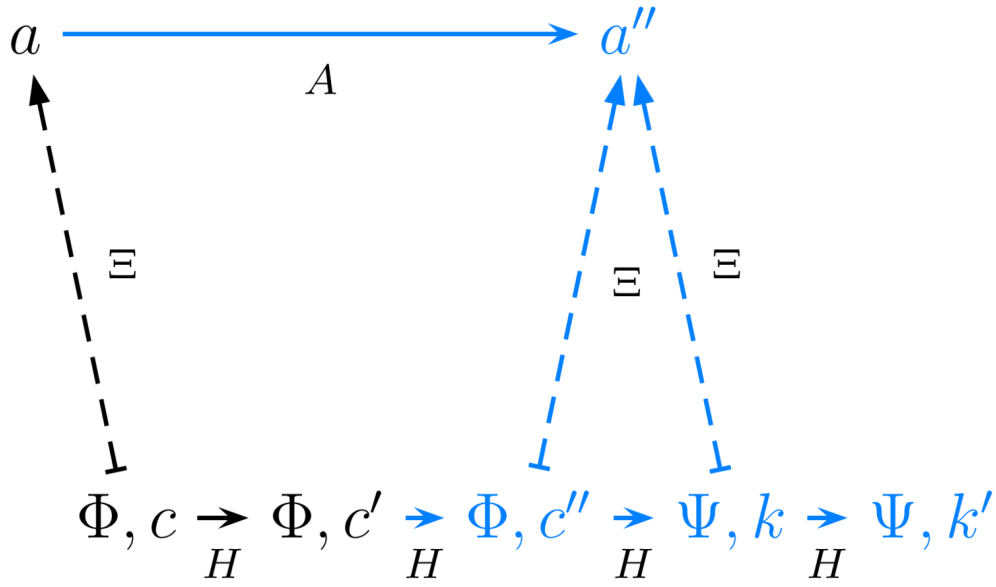
Migration (Optimized)



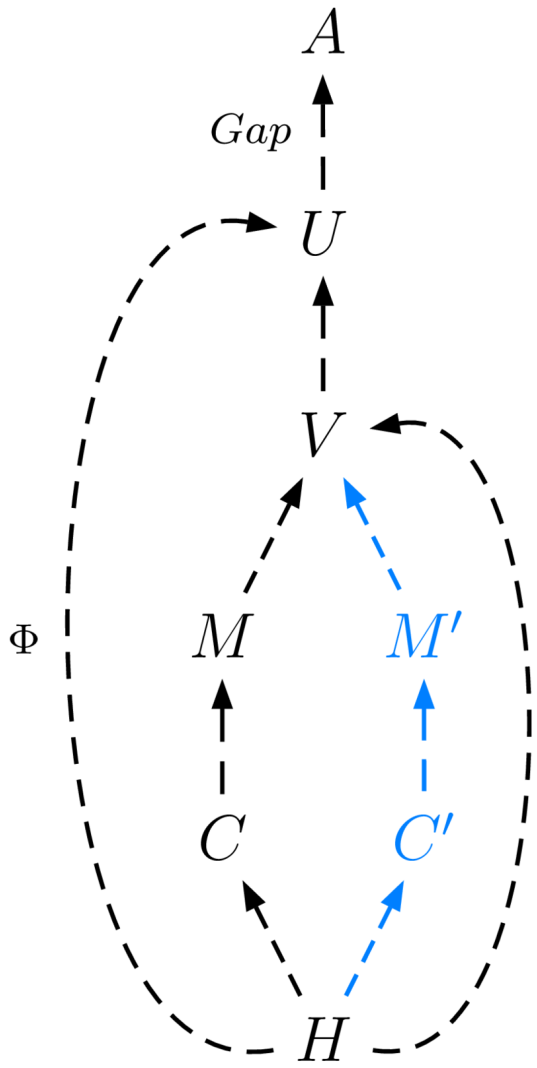
Migration (Implemented)



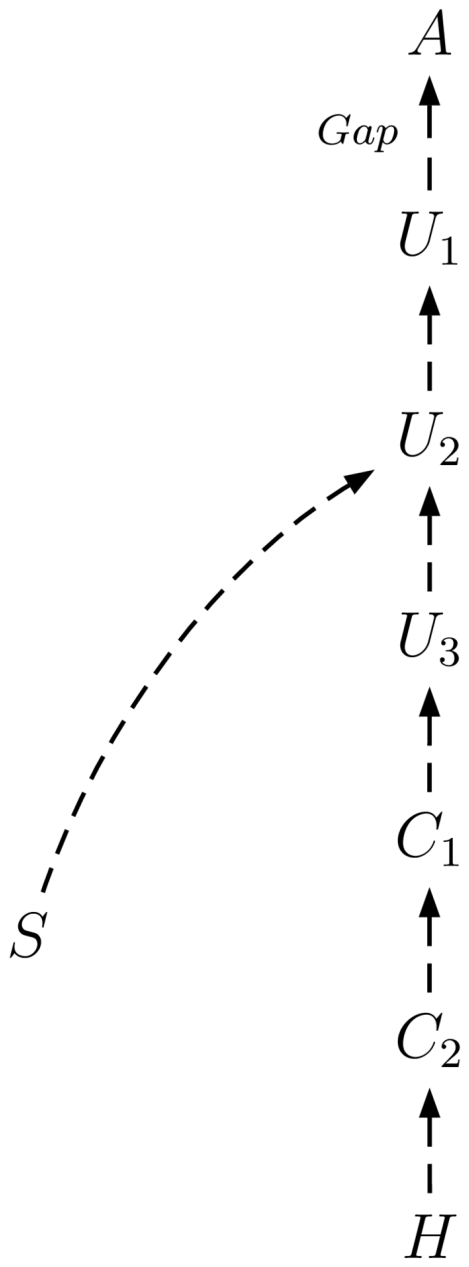
Migration (Factored out)



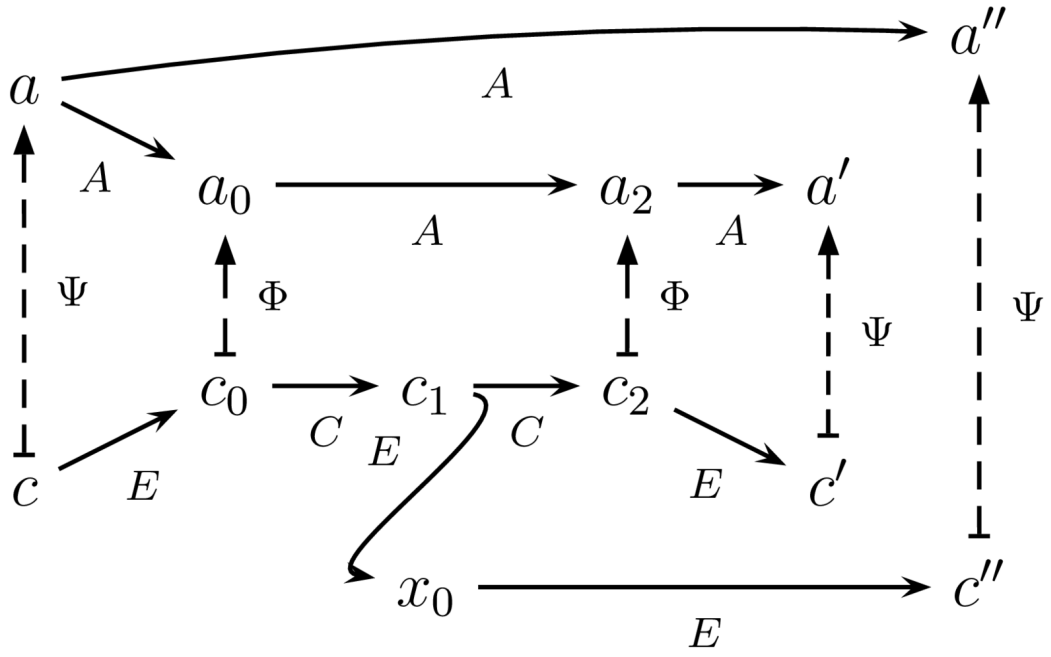
Migration Tower



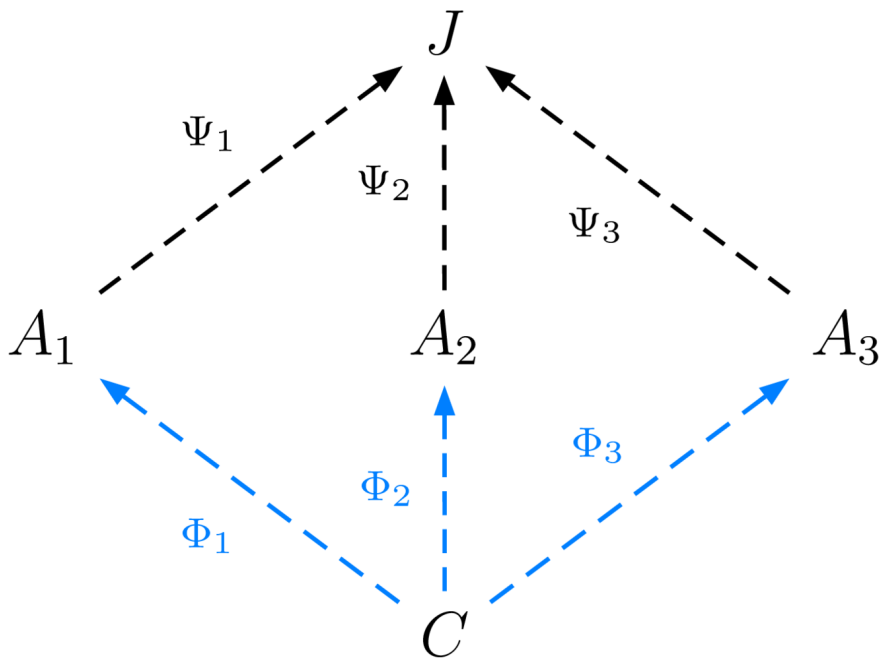
Semantic Tower



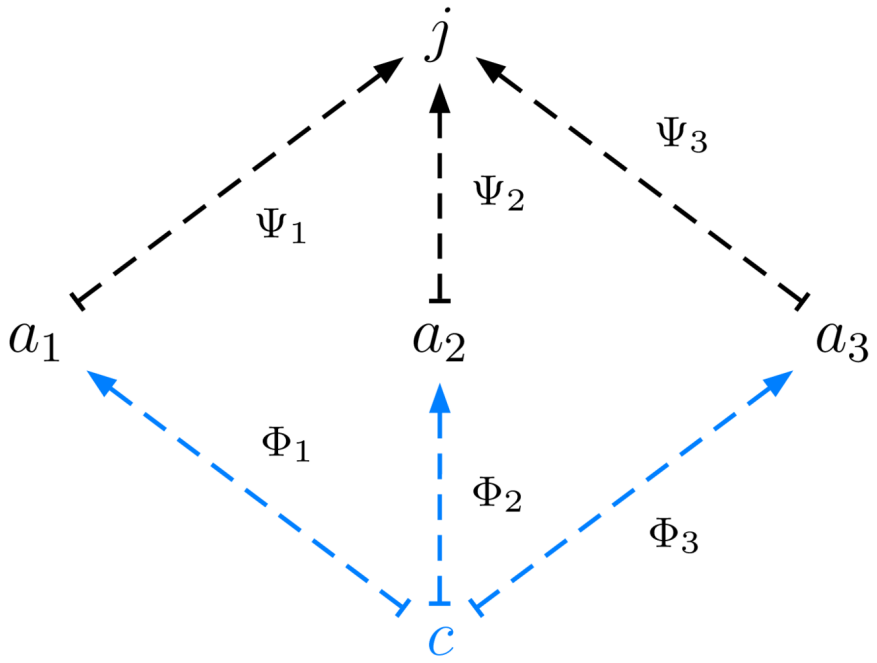
Optimistic Evaluation



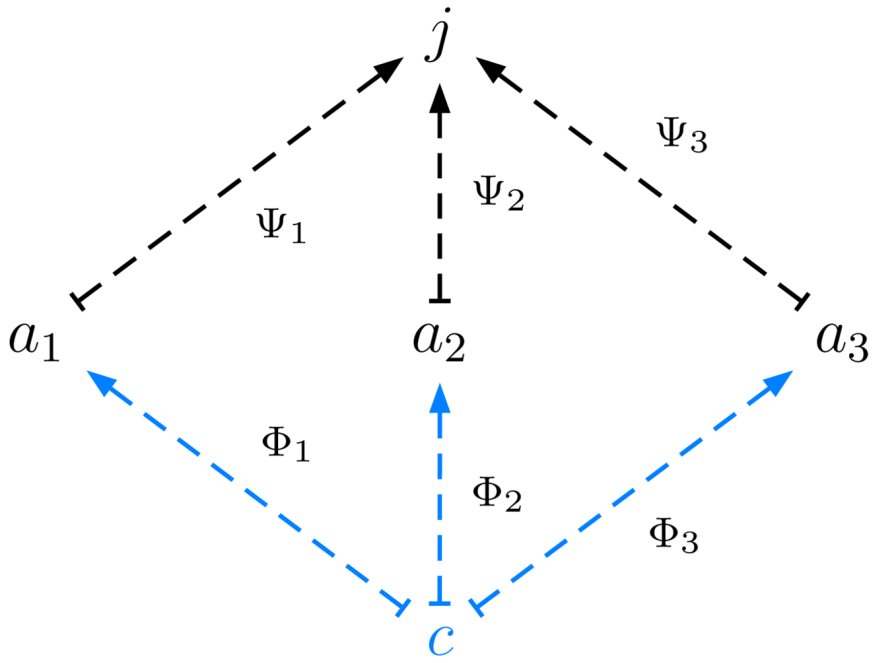
Aspect-Oriented Programming (1)



Aspect-Oriented Programming (2)



Aspect-Oriented Programming (2)



Constraint Logic Meta-programming!

Natural Transformations of Implementations

Natural Transformations of Implementations

Automatic Instrumentation

Code and Data Coverage

Omniscient debugging

Resource Accounting

Parallelization

Orthogonal persistence

Virtualization

etc.

IV. Runtime Meta-programming Architecture

Runtime Architecture

Runtime Architecture

Development Platform (Emacs, IDE, ...)

User Interface Shell

Operating System

Distributed and Virtualized Application Management

Every Program has a Semantic Tower

Semantics on top + Turtles all the way to the bottom

Top specified by User, bottom controlled by System

For the PLs your build, those you use

Static or dynamic control

Every Tower has its Controller

Runtime Meta-program, Shared (or not)

Virtualization: control effects, connect I/O

Reflective Tower of Meta-programs

Another dimension to diagrams! Turtles?

Implicit I/O

```
Input :: tag -> IO indata
Output :: tag -> outdata -> IO ()
```

Handled by controller

Virtualization of effects at language level

Dynamically reconfigurable

Performance: Dynamic Global Optimization

When configuration changes, migrate

Optimize the current configuration

Minimize encoding, Zero copy

Skip unobserved computations

Simplicity: Separate program and metaprogram

Example: File selector, UI, etc.

Evolve, Distribute, Share, Configure separately

Separate Capabilities, Semantics

Robustness, Security: Smaller Attack Surface

Not Just a Library

Semantic separation vs inclusion

Bound at Runtime vs Fixed at Compile-/Load- time

Different scopes and capabilities

Different control flow

Related Works and Opportunities

Formal Methods for proving program correctness

Open Implementation, AOP...

Many hacks for GC, Migration, Persistence...

Virtualization, distribution...

Common Theme

Programming in the Large, not in the Small

Software Architecture that Scales

Semantics matter

Dimensions of Modularity beyond the usual

The Take Home Points (redux)

Implementation is co-(Abstract Interpretation)

Safe points are a key concept

First-class: the opposite of magic

First-class safe points (= PCLSRing!)

Applications: Migration, Optimistic Evaluation, etc.

Composing implementations for fun and profit

Runtime meta-programming brings new modularity

Challenge

Put First-class Implementations in your platform

Platform: PL, IDE, OS, Shell, Distributed System

Factor your software into meta-levels

Enjoy simplification, robustness, security

The Meta-Story

My contribution is mostly not technical.

It is more ambitious:

The Meta-Story

My contribution is mostly not technical.

It is more ambitious:

A change of point of view about computing

The Meta-Story

My contribution is mostly not technical.

It is more ambitious:

A change of point of view about computing

Thank you!

My blog: *Houyhnhnm Computing*

http://ngnghm.github.io/