First-Class Implementations

Climbing Up the Semantic Tower — At Runtime

François-René Rideau, *TUNES Project*

DeepSpec Summer School, 2018-07-23

http://fare.tunes.org/files/cs/fci-ds2018.pdf

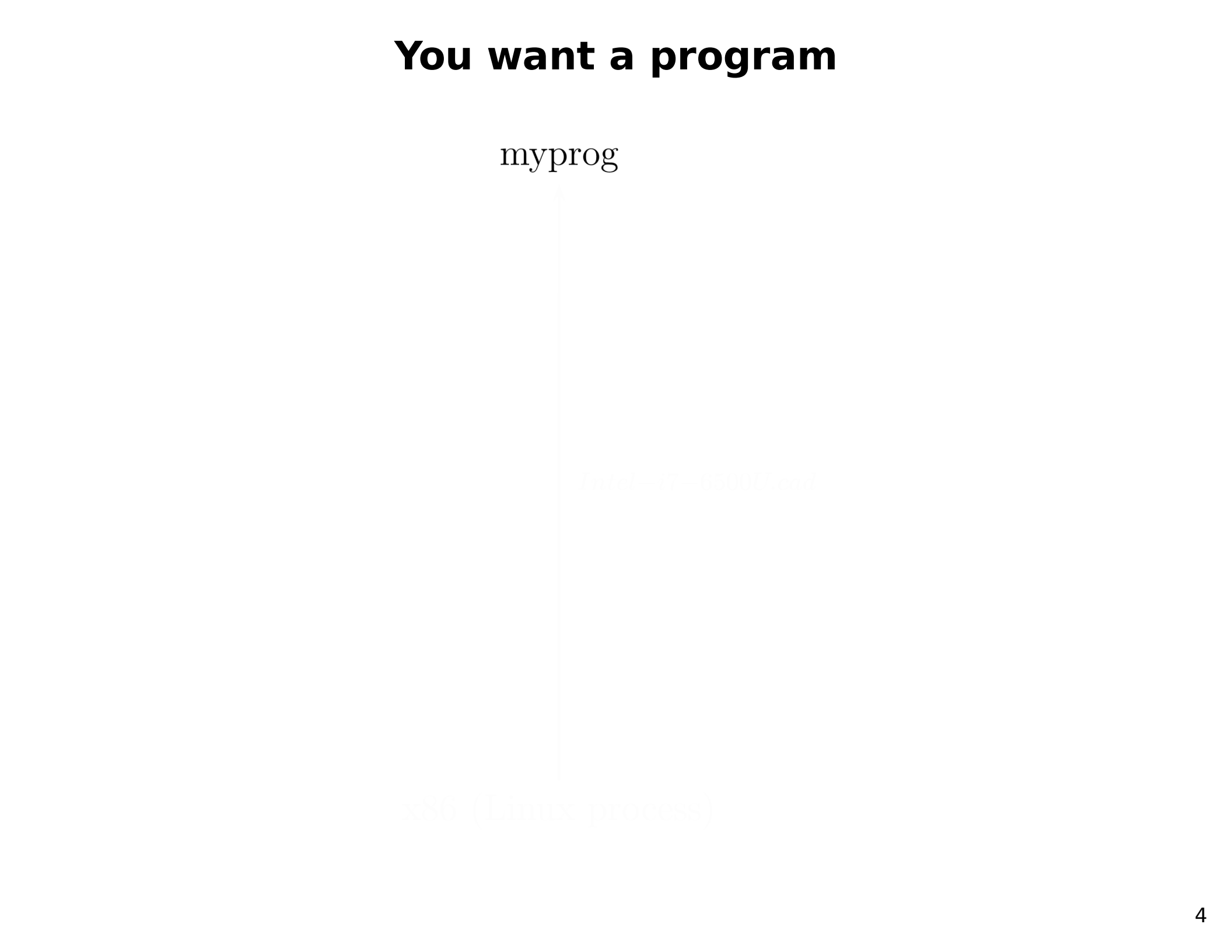
Based on my PhD thesis (completed in 2017, not defended)

Also showed at OBT 2018, LambdaConf 2018

I. Implementations

I.1 A Universal Framework

You want a program

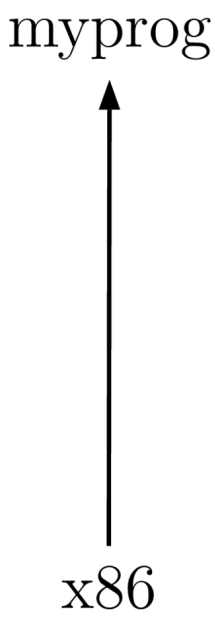


You have a PC

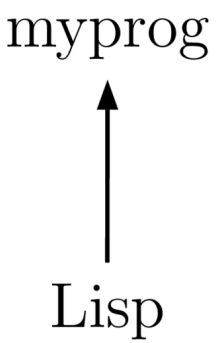
Intel-i7-6500U Load

x86 (Linux process)

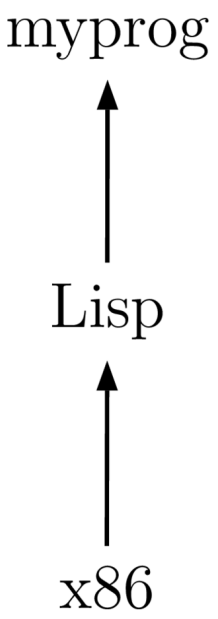
You write an implementation



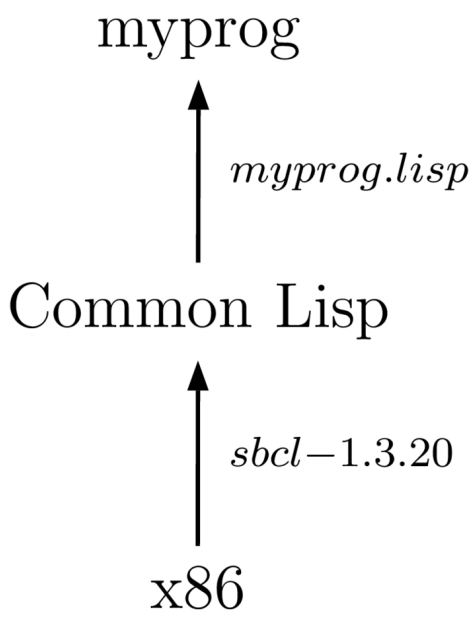
In the best possible language



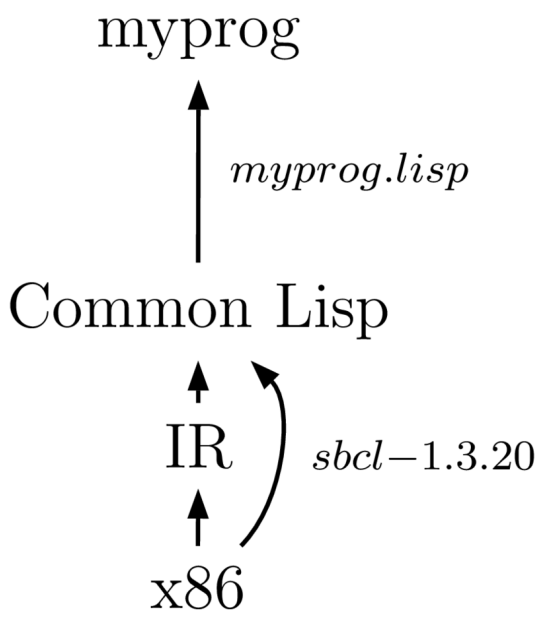
The language itself has an implementation



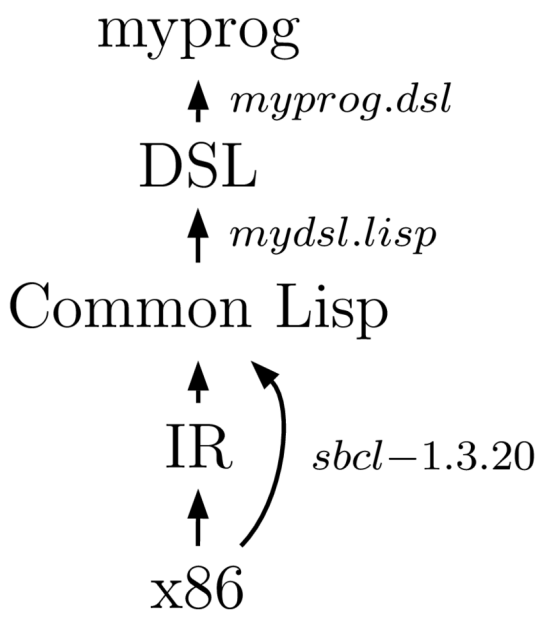
Specific dialects, implementations, versions...



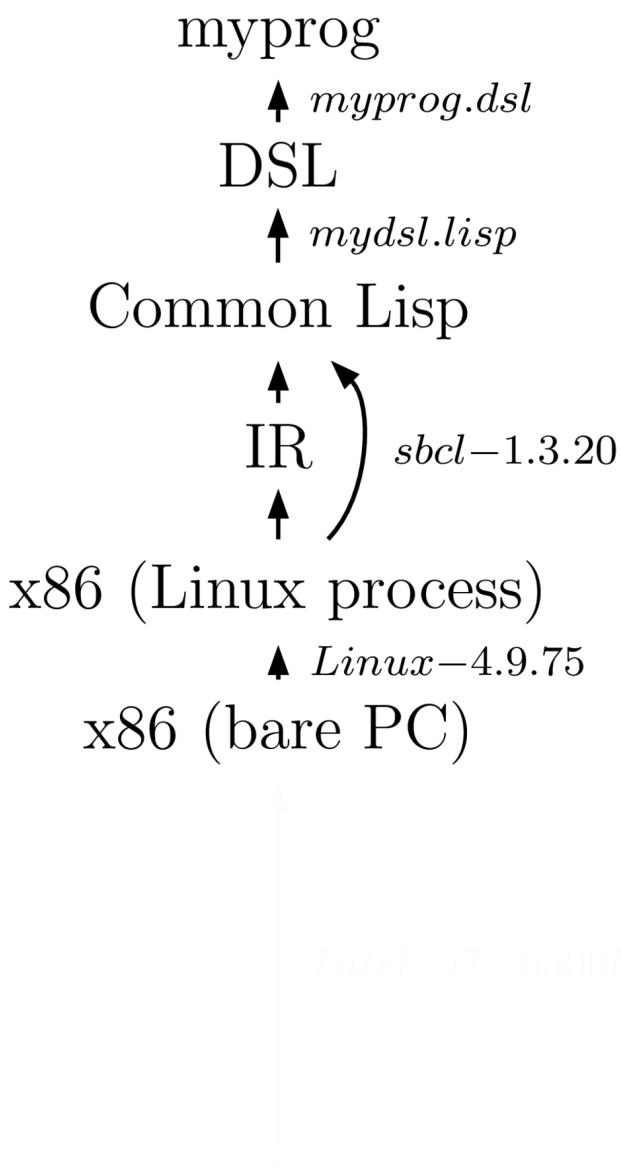
Compiling is hard, use an IR...



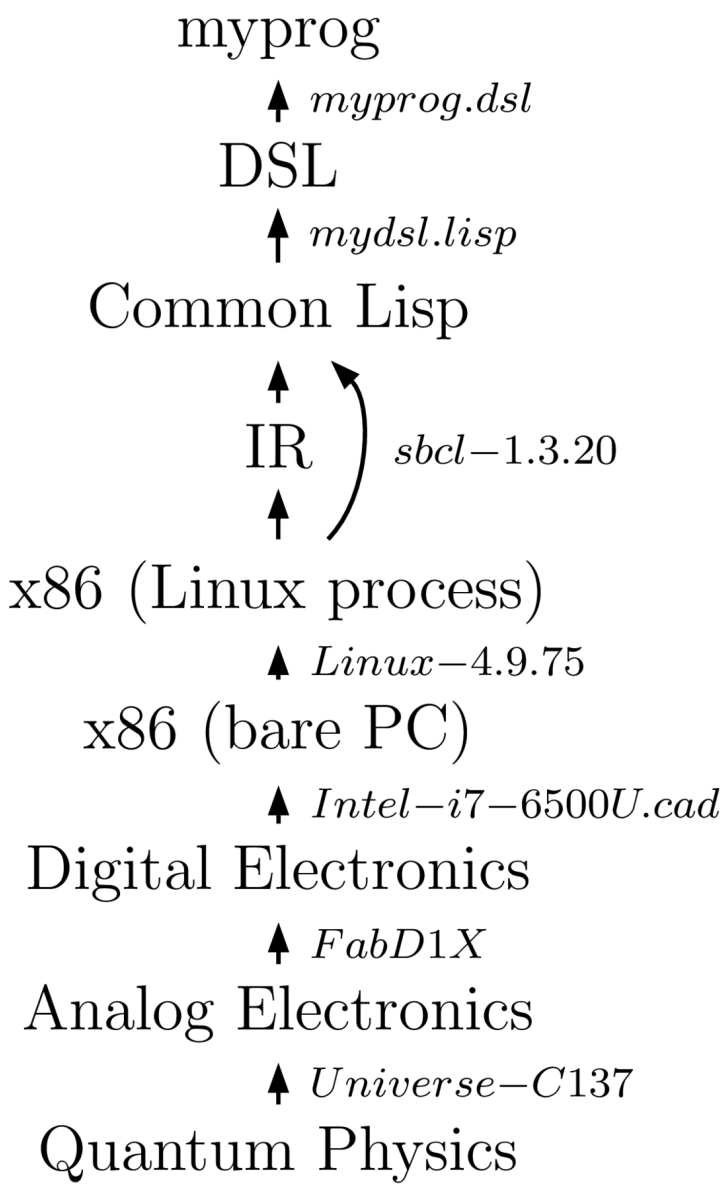
Programming is hard, use a DSL...



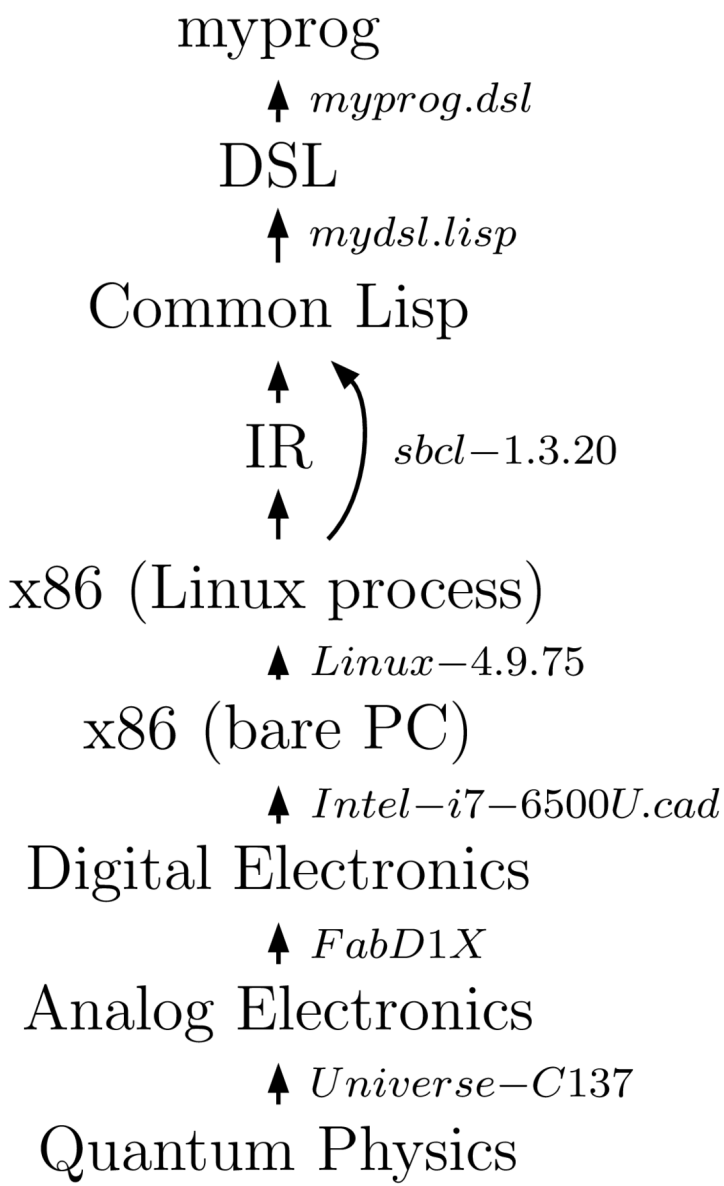
What do you mean, x86?



There is no bottom!



Always finer divisions



Existing Semantic Formalisms to Unify

Operational Semantics (Small Step)

Operational Semantics (Big Step)

Labeled Transition Systems

Term Rewriting, Rewrite Logic

Modal Logic, Hoare Logic, Refinement

Partial Order

Abstract State Machines

Denotational Semantics reducing to the above

Denotational Semantics with equational theory

Category Theory

Universal: graphs, preorders, labeled transitions...

Simple core: nodes, arrows, structure preservation

Unlimited abstraction: always higher categories

Structural theorems "for free"

Types, Curry-Howard Isomorphism

Seeking the essential: no incidental punning

Computation as Categories

Nodes: states of the computation

Arrows: transitions between states, traces

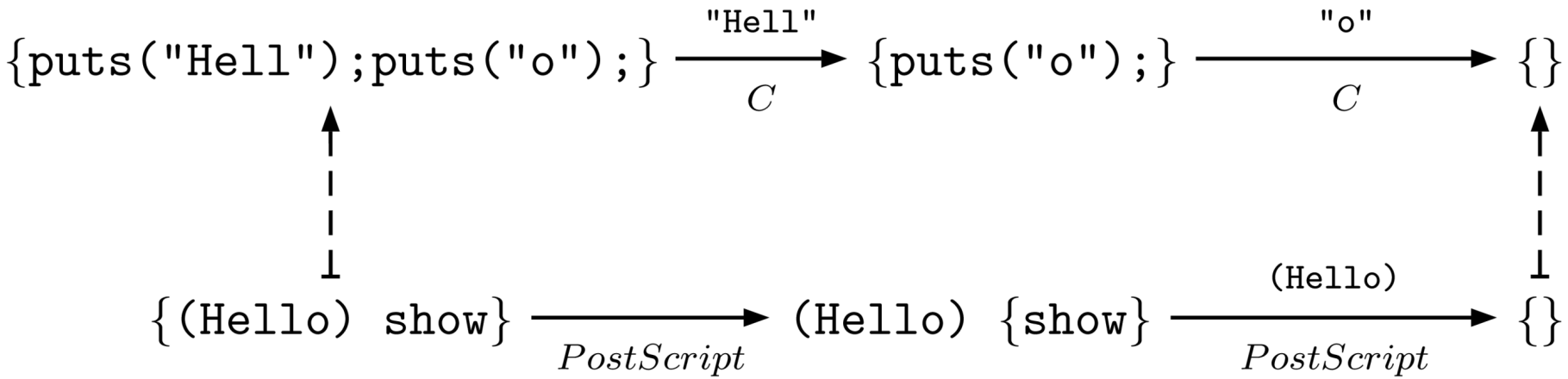
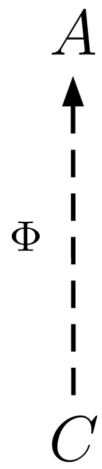


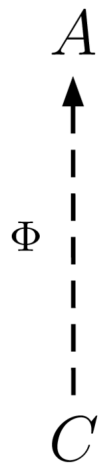
Figure conventions:

- Computation progresses left to right
- Effect label above, category (subset) below
- Dotted lines for partiality and other effects

(Abstract) Interpretation

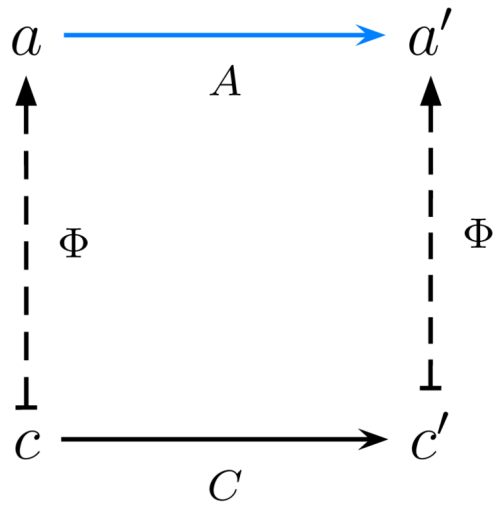


(Concrete) Implementation



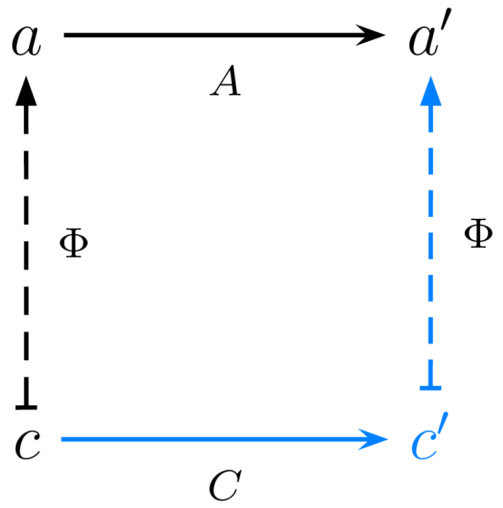
I.2 Properties of Implementations

Soundness

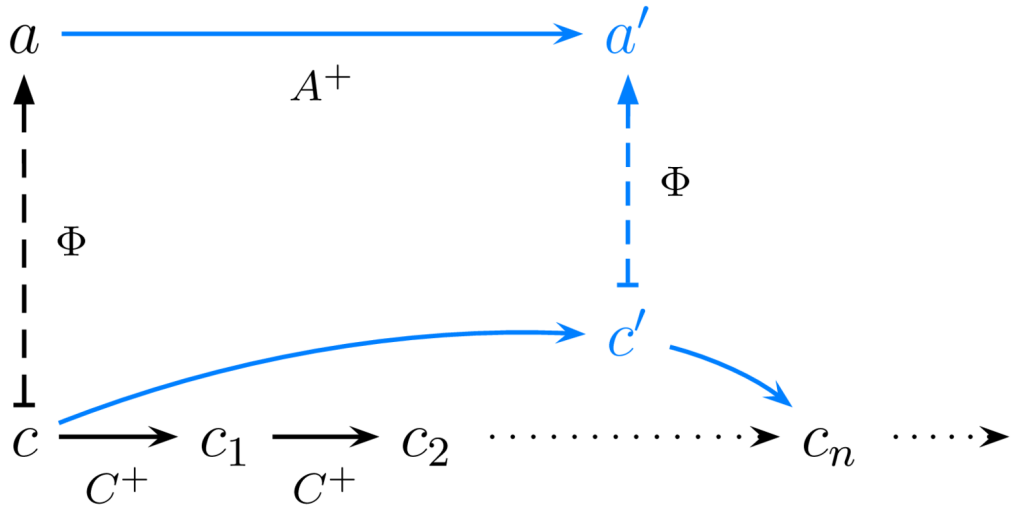


Totality

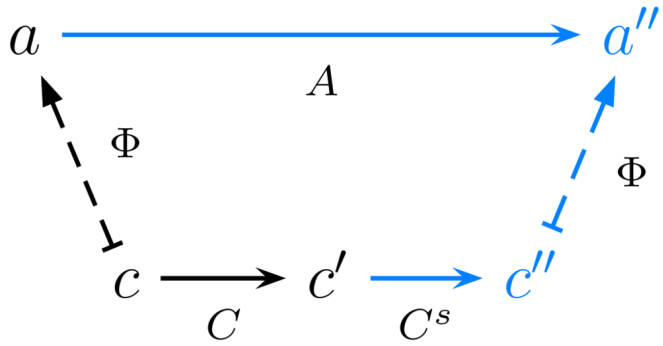
Completeness



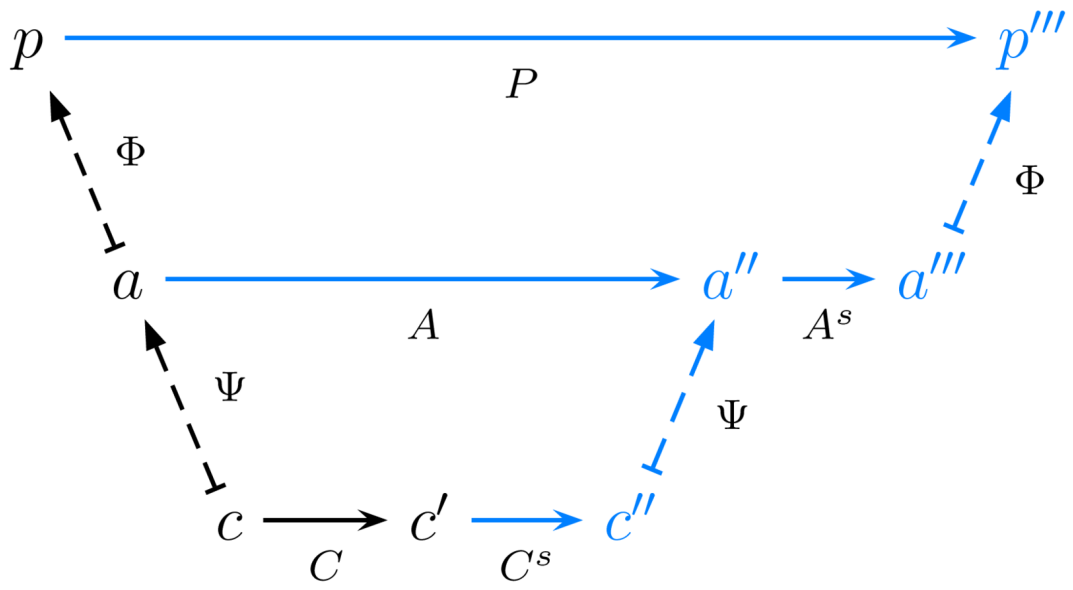
Liveness



Observability (aka PCLSRing)

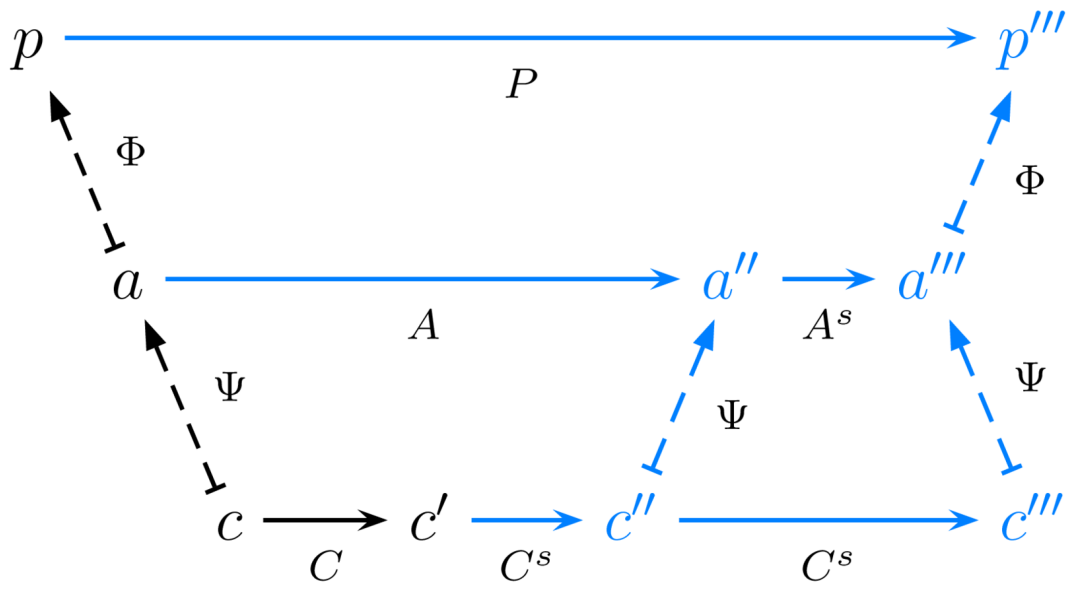


Observability (aka PCLSRing)



... not composable!

Observability + Completeness



Composable!

II. First-class Implementations

II.1 Protocol Extraction

Protocol: Categories (in Agda)

```
record Category … : Set … where …
  field
    Obj : Set …
    _⇒_ : Rel Obj …
    id : ∀ {A} → (A ⇒ A)
    _∘_ : ∀ {A B C} → (B ⇒ C) → (A ⇒ B) → (A ⇒ C)
  …
```

Showing fields with computational content

Many more fields for logical specification

Protocol: Categories (in Haskell)

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

Pure total functions: →

Effectful functions: ⇢ (partial, non-det…)

Protocol: Operational Semantics

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

Protocol: Implementation

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

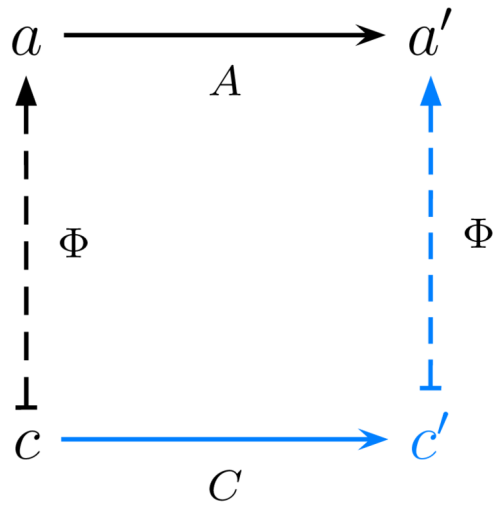
So far, a (partial) functor from `c` to `a`

Arr = pirate sound = functorial map

Protocol: Totality

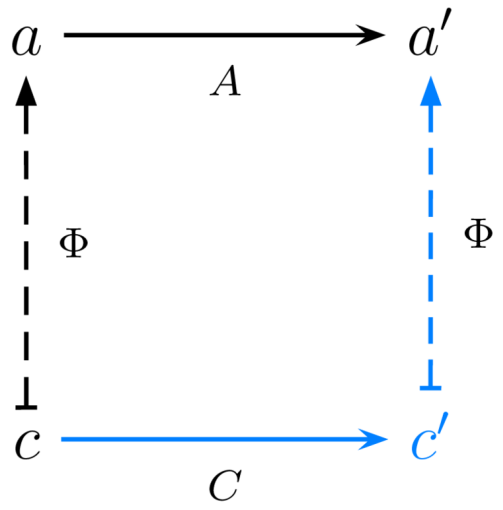

```
implement :: a ⇢ c
```

Protocol: Completeness



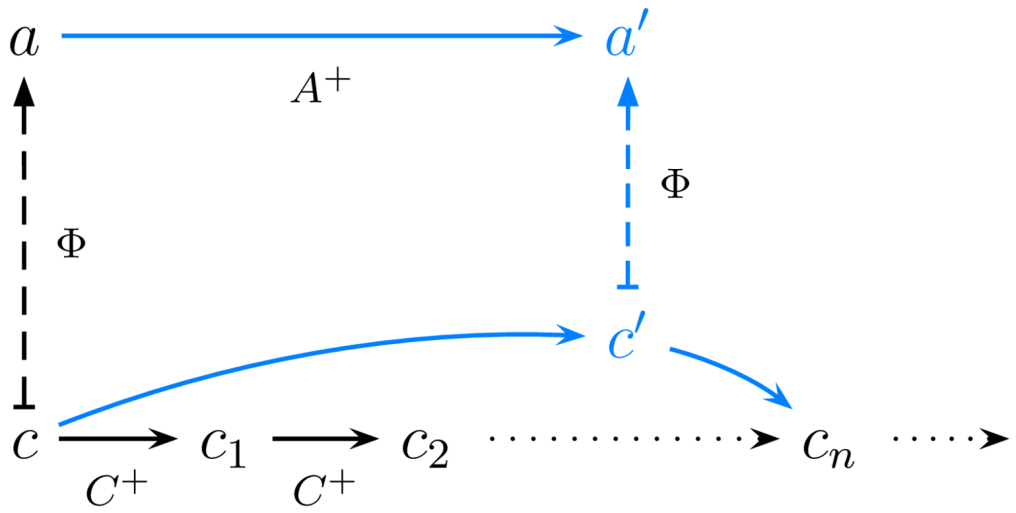

```
implementArr :: c → (Arr a) ⇢ (Arr c)
```

Protocol: Completeness (with Dependent Types)



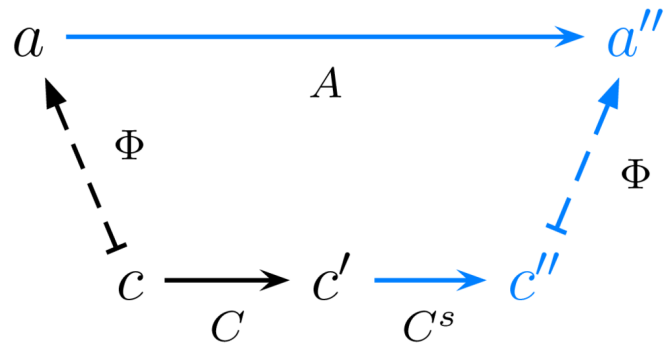

```
implementArr :: c ⟶ (Arr a) ⇢ (Arr c)

implement⇒ : ∀ (c : C.o) {a a' : A.o}
(f : C.⇒ a a') {Φ.o c a} → ∃(λ {c' : C.o} →
∃(λ (g : C.⇒ c c') → Φ.⇒ g f))
```

Protocol: Liveness




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

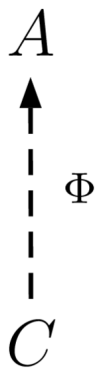
Protocol: Observability (PCLSRing)




```
safePoint :: c ⇢ Arr c

safeArrow :: Arr c ⇢ Arr c
```

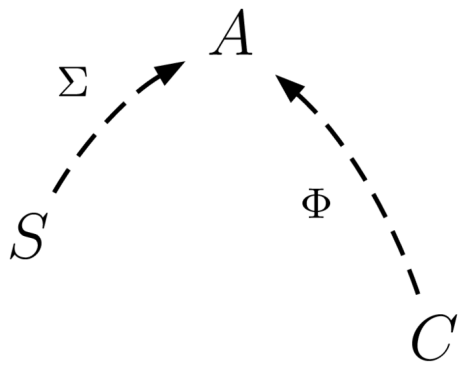
II.2 The Semantic Tower

Compilation (1)



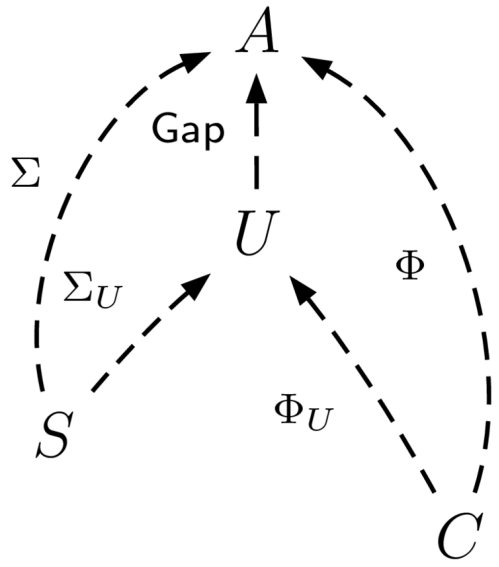

```
implement :: (Impl a c) ⇒ a ⇢ c
```

Compilation (2)



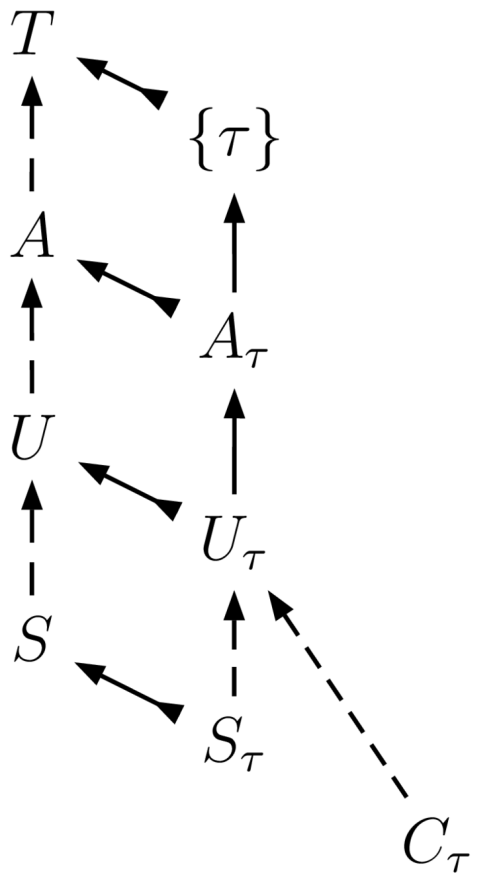

```
interpret :: (Impl a s) ⇒ s ⇢ a
implement :: (Impl a c) ⇒ a ⇢ c
```

Compilation (3)



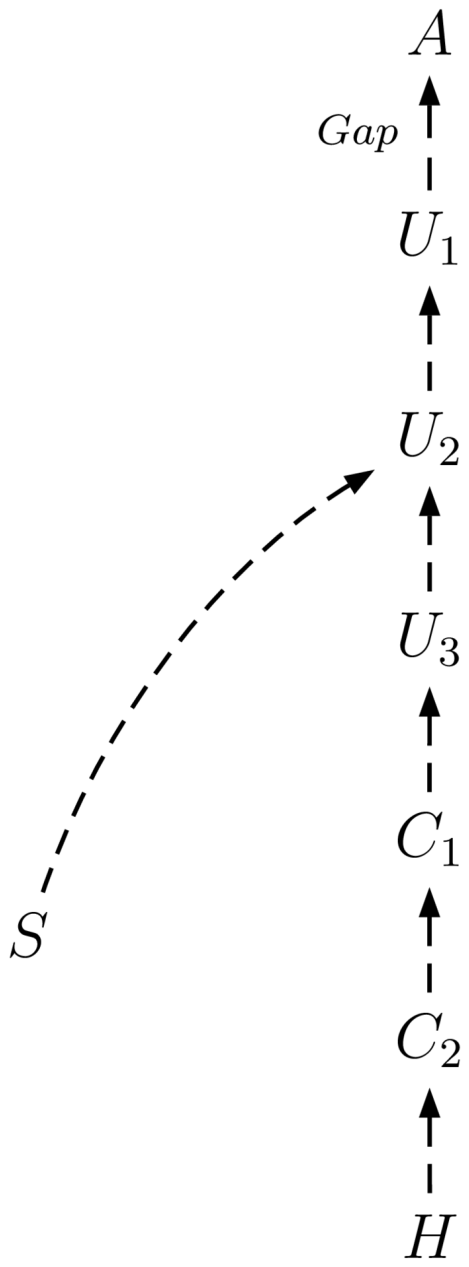

```
u :: OpSem -- specify up to what rewrites
interpret :: (Impl u s) ⇒ s ⇢ u
implement :: (Impl u c) ⇒ u ⇢ c
```

Static Type Systems

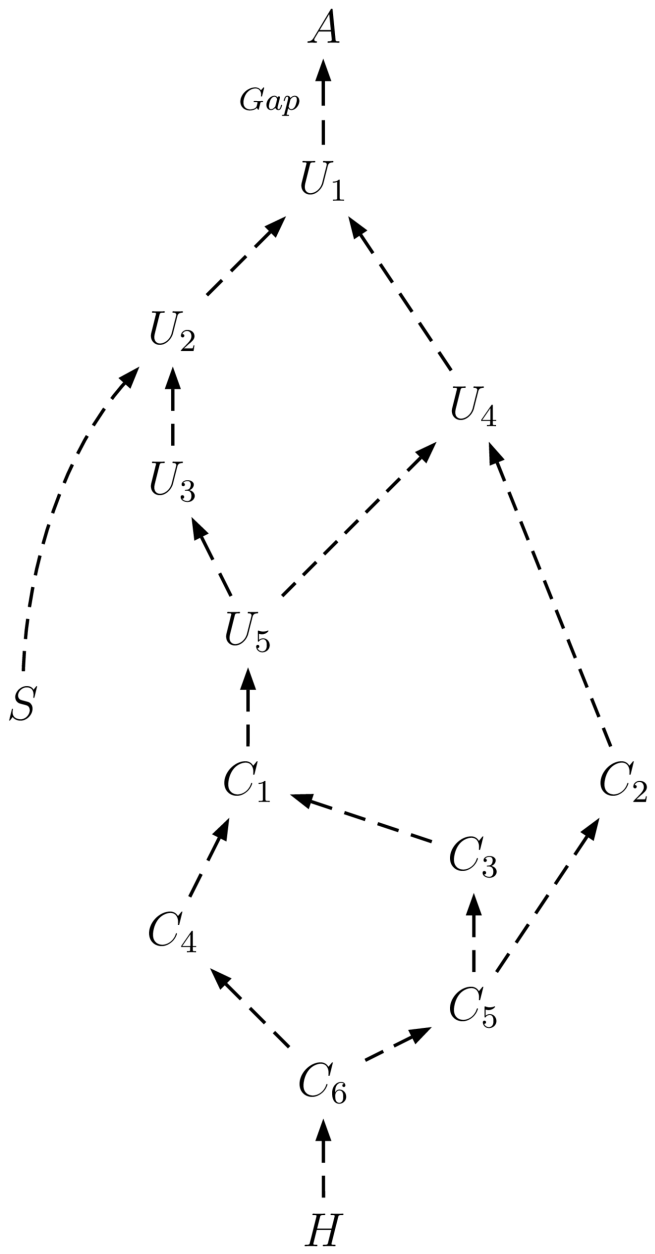


Subject reduction: *T* contains no exomorphisms

Semantic Tower



The Tower is not Linear



III. Applications

III.1 Reconciling Reflection and Semantics

More reinterpretations...

Aspect-Oriented Programming

Erlang-style Fault Tolerance

Developing, Refactoring

Migration: Mobility, GC, JIT, Zero Copy, Reconfig...

Dynamic Control: implicit IO, virtualization...

Reflective Towers: Turtles all the way down...

Interaction: Operating System Shell, IDE, ...

Architecture: Meta-Objects, not reducible to libraries

Fruitful change in Perspective

Correctness

Composability

Runtime Optimization

Retroactive Features

Predictable Cost-Reduction

Requirement: Full Abstraction

Computations have a clear opaque bottom:

- It’s perfectly clear what the bottom is

- The bottom is totally opaque

Indeed, what’s below can change at runtime!

Alternatively, include what’s "below" in the spec

Needed: explicit language or system support

III.2 Natural Transformations of Implementations

Instrumentation

Tracing, Logging, Stepping, Profiling

Omniscient debugging, Comparative Debugging

Code and Data Coverage

Resource Accounting, Access Control

Parallelization, Optimistic Evaluation

Orthogonal persistence

Virtualization

Optimizations

Natural Transformation

Twist: *dual* of nat. transf. on *dual* of (partial) funct.

Adjunction: forget details added by the instrumentation

Automatic Instrumentation

Universal transformations

Composable transformations

Amenable to formal reasoning

Open problem, but promiseful approach

Conclusion

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

Reason about Implementations: Category Theory!

Observability: Key neglected concept — safe points

Practical Protocol Extraction: First-Class Impl.

Explore the Semantic Tower — at runtime!

Principled Applications: Migration, etc.

Natural Transformations generalize Instrumentation

Runtime Reflection *and* Static Semantics

Price: Full Abstraction, Observability, Interpretation

Challenge

Put First-class Implementations in your platform

Factor your software into meta-levels

Develop Generic Tooling, Reflective Architecture

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

My thesis (undefended):
https://bit.ly/FarePhD

My blog: https://ngnghm.github.io/

Ancient: https://tunes.org/

Legicash is hiring Coq developers

Legicash is looking for Academic Collaborators