Imperial College London

Algebraic Effects Meet Hoare Logic

in Cubical Aqda

Donnacha Oisín Kidney Zhixuan Yang

Nicolas Wu

POPL 2024, London

This paper, technically

- 1. Formalise **algebraic effects** (on h-sets) in Cubical Agda
- 2. Define a generic **Hoare logic** on top of program equalities
- 3. Prove an interesting theorem



This paper, technically

- 1. Formalise **algebraic effects** (on h-sets) in Cubical Agda
- 2. Define a generic **Hoare logic** on top of program equalities
- 3. Prove an interesting theorem

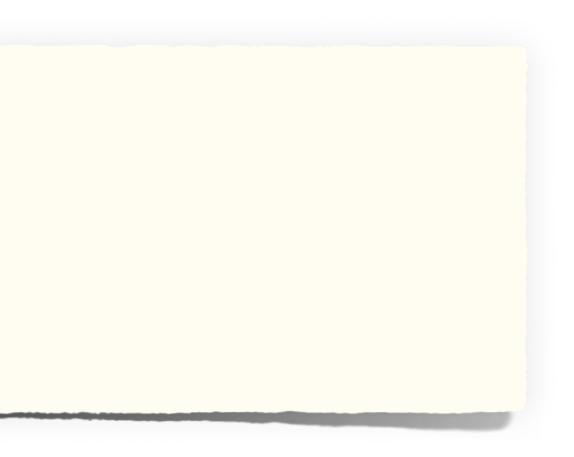
But why?



Language-formalisation checklist

G Formalise the syntax

OFormalise the semantics

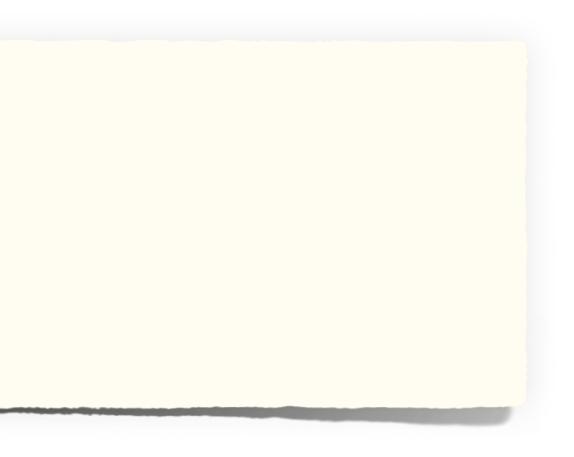


Language-formalisation checklist



OFormalise the semantics

What is "semantics"?

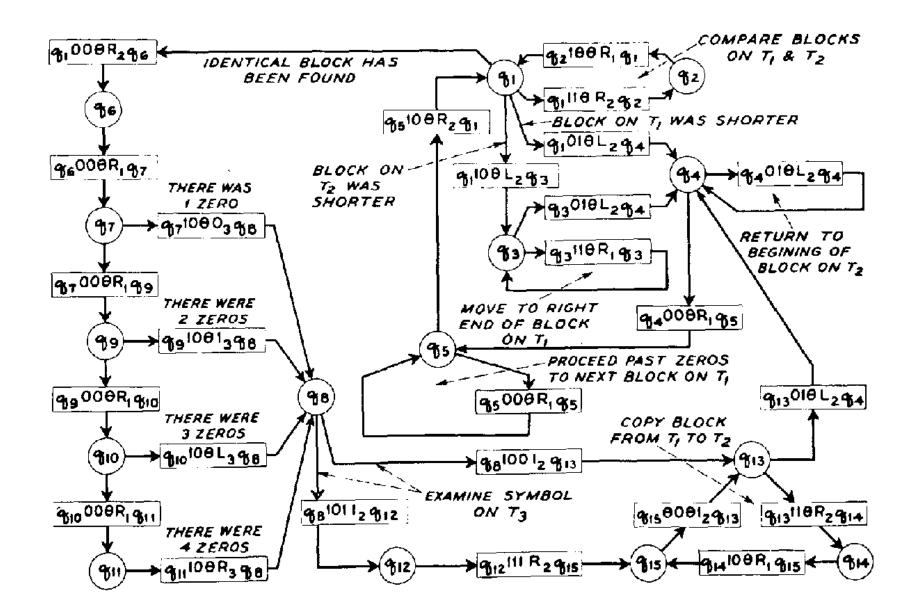


Operational semantics

A programming language is treated as an abstract (symbolic) machine:

- Terms are states
- Computing is transitioning

just like a Turing machine



A universal Turing machine by Moore (1952) which fits into one picture

But what is a language

In its ordinary sense, *languages* are *systems for describing things*

$\llbracket a \text{ cat in } a \text{ bag } \rrbracket =$





CC-BY-NC-ND 3.0 by ClassicSonicSatAm

But what is a language

In its ordinary sense, *languages* are *systems for describing things*

$\llbracket a \text{ cat in } a \text{ bag } \rrbracket =$





CC-BY-NC-ND 3.0 by ClassicSonicSatAm

But what is a language

In its ordinary sense, *languages* are *systems for describing things*

[a cat in a bag] =

It's not always clear from the operational semantics what the language describes



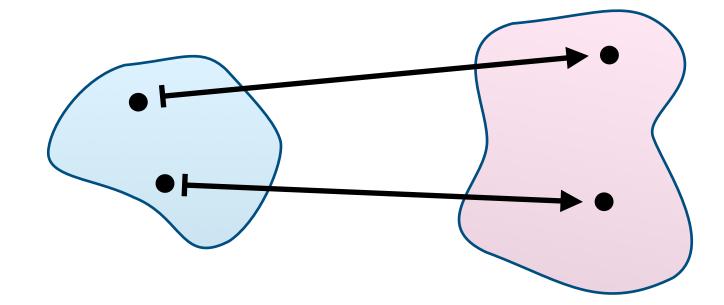


CC-BY-NC-ND 3.0 by ClassicSonicSatAm

Why we design languages

- Most PLs nowadays are designed for describing certain things easily:
 - 1. *some mathematical concepts* (sets, domains, sheaves, ∞ -groupoids, quasi-Borel space...)

2. terms up to some operational property (logical predicates/relations, contextual equivalence, applicative bisimilarity...)



$\{ \Gamma \vdash t : \tau \mid t \text{ blah blah in the } \}$ operational semantics }

$$\{\Gamma \vdash t : \tau\} / \cong_{obs}$$



Checklist (ver. 2)

Formalise the syntax

Formalise the "intended" denotational model (which may be based on operational semantics)

Checklist (ver. 2)

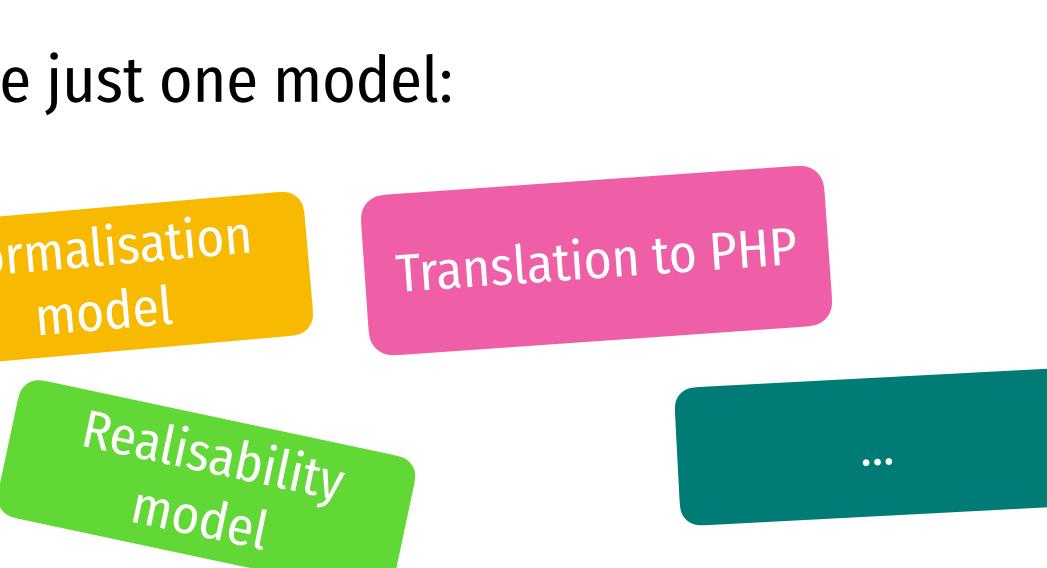
Formalise the syntax

Formalise the "intended" denotational model (which may be based on operational semantics)

But there is no reason to have just one model:

Set-theoretic model

Normalisation mode



Checklist (final)

Formalise the syntax of the language

Formalise the general notion of models

Define the models that we are interested in

Programming and reasoning model-independently if possible

This can be called the *logical approach* to program verification.

This paper, conceptually

A framework for formalising first-order languages:

- The user specifies a language by *operations* and *equational axioms*
- The library provides
 - *models*: a set implementing all operations
 - *free models*: syntactic terms *quotiented* by equations
 - *reasoning tools* for free models



An example

A language for nondeterministic parsing is specified by

- Operations
 - get, put for accessing the token stream
 - **or, fail** for nondeterministic branching
- Axioms
 - These two groups of operations commute with each other
 - Some standard equations on each of them, e.g. or(x, or(y, z)) = or(or(x, y), z)
- Models: free models, String -> Bag A, terms up to bisimilarity, ...

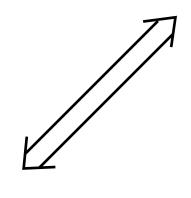
An example

Parsers can be defined as elements in the free model (Term below).

A fragment of a parser of binary trees with leaves \diamondsuit and \blacklozenge :

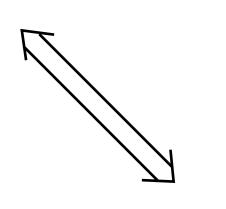
Equational Reasoning

For two elements s and t in the free model,



s can be rewritten to t by the equational axioms

- s equals t



s and t have equal meanings in all models



In principle, from the equational axioms, we can show that

(do push (print t); parse-tree n) = return t

for every tree t and sufficiently large n.



In principle, from the equational axioms, we can show that

(do push (print t); parse-tree n) = return t

for every tree t and sufficiently large n.

A direct equational proof is painful...

Hoare-style reasoning

A generic Hoare logic by Schröder and Mossakowski (2003) is formalised:

- Assertions P, Q are programs in the free model returning (h-)propositions Ω
- The Hoare triple $\{P\} x \leftarrow t \{Q x\}$ is encoded as an equality:

do
$$a \leftarrow P$$

 $x \leftarrow t$
 $b \leftarrow Q x$
return (x, $a \Rightarrow b$)



Connecting the two worlds

- A proposition established using Hoare logic can be used in equational reasoning
- **Main theorem.** Given {} $x \leftarrow t$ {return $\phi(x)$ } and $\forall x . \phi(x) \rightarrow f(x) = g(x)$ then ($x \leftarrow t$; f(x)) = ($x \leftarrow t$; g(x))
- Proof (classically). easy
- Proof (constructively). surprisingly hard and bizarre...
- **Usage**: turning tedious equational proofs to more intuitive Hoare-style proofs.

An example

It is easier to first show a Hoare triple

{remaining (print t ++ r)} t' \leftarrow parse-tree n {return (t' = t) \land remaining r}

from which we can derive the earlier goal (do push (print t); parse-tree n) = return t.



- (Most) programming languages are both languages and machines
- Hoare-style reasoning and equational reasoning complement each other



Chank you