Representing and Manipulating Contexts

A Tool for Operational Reasoning

David Sands

Chalmers University of Technology

SOS Workshop,

London, 30 August 2004

Acknowledgements: Swedish foundation for Strategic Research

Definition 1.0

Definition 1.0

An Old-Fashioned Course in Semantics

Aim

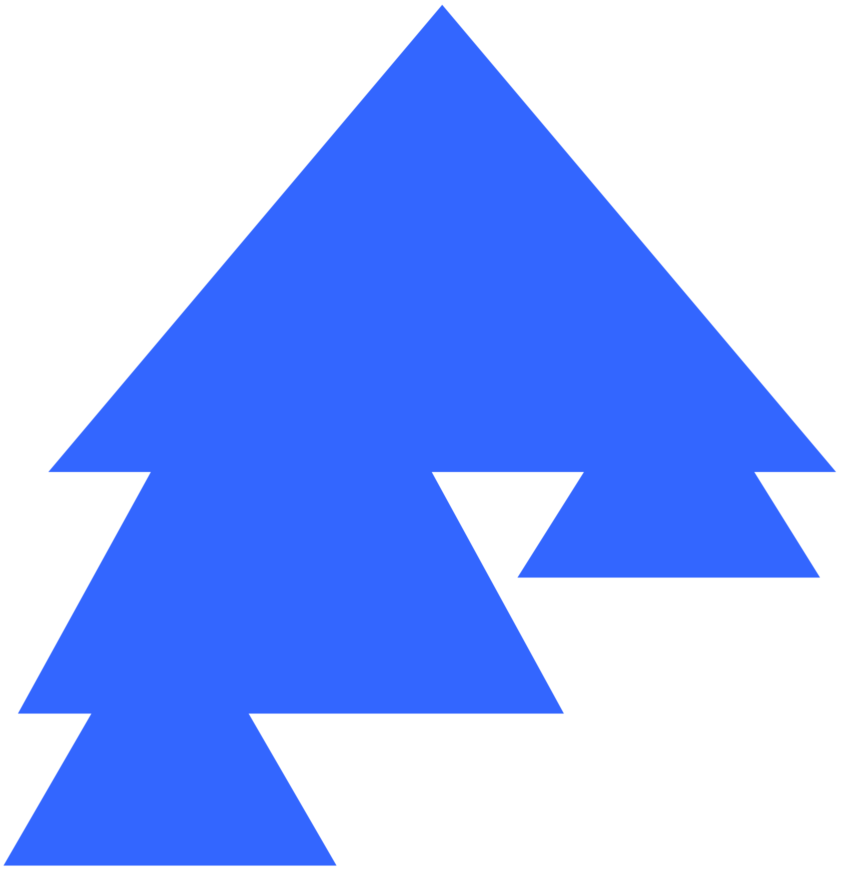
Introduce a useful technique for reasoning about higher-level properties of programming languages from simple operational semantics

Lifting computation rules from terms to contexts

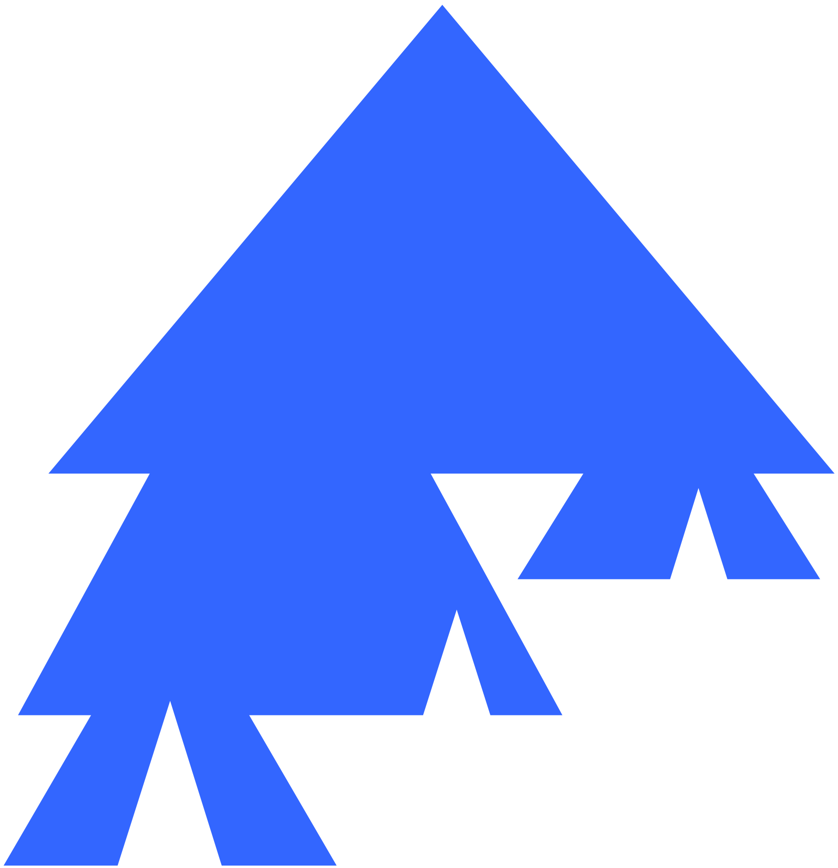
Contexts: informal notation

- C[ ] denotes a context – a program phrase containing zero or more missing subphrases.
- C[M] denotes the program phrase obtained by plugging M into the holes
- Not the same as substitution

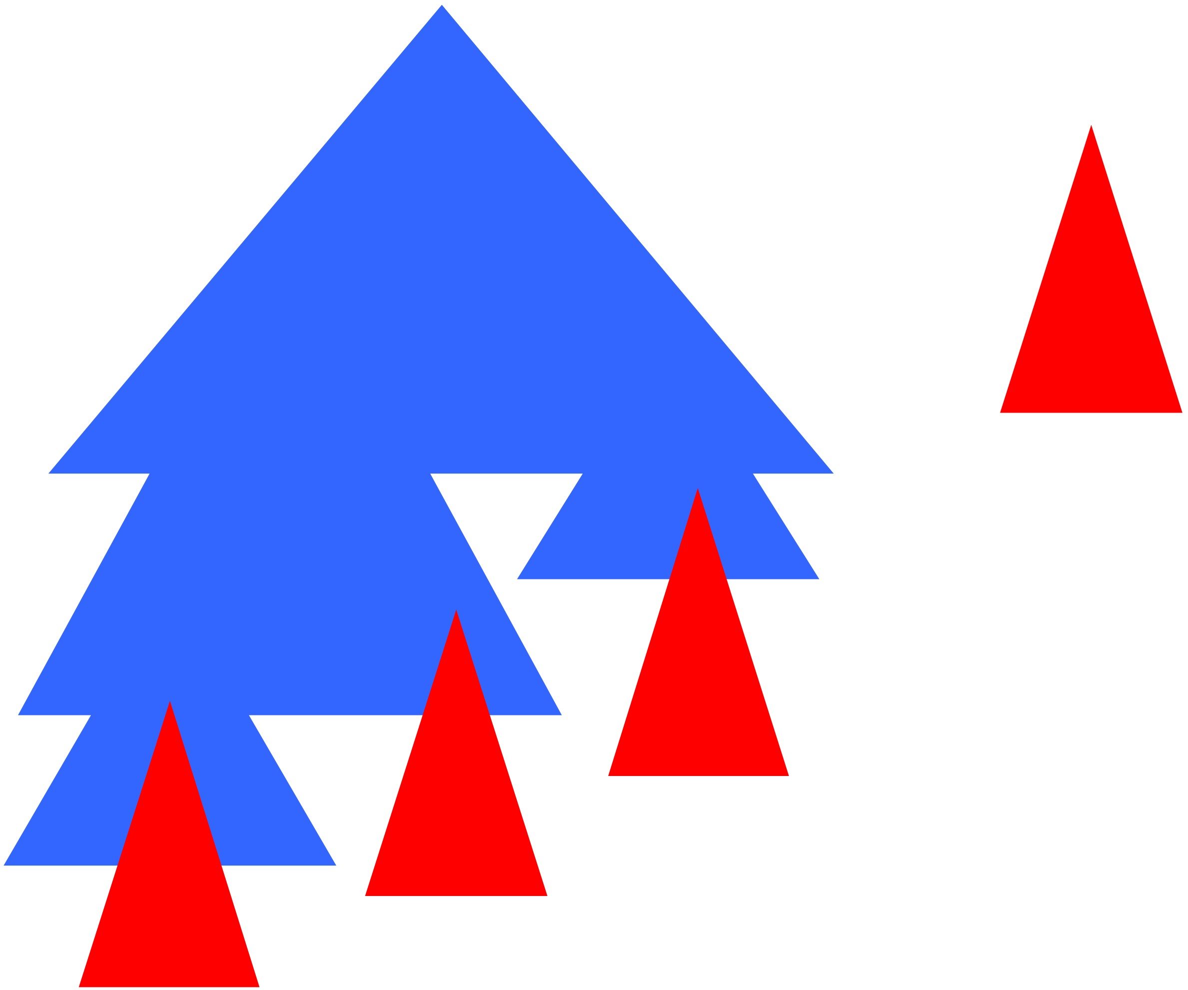
A Syntactic Term



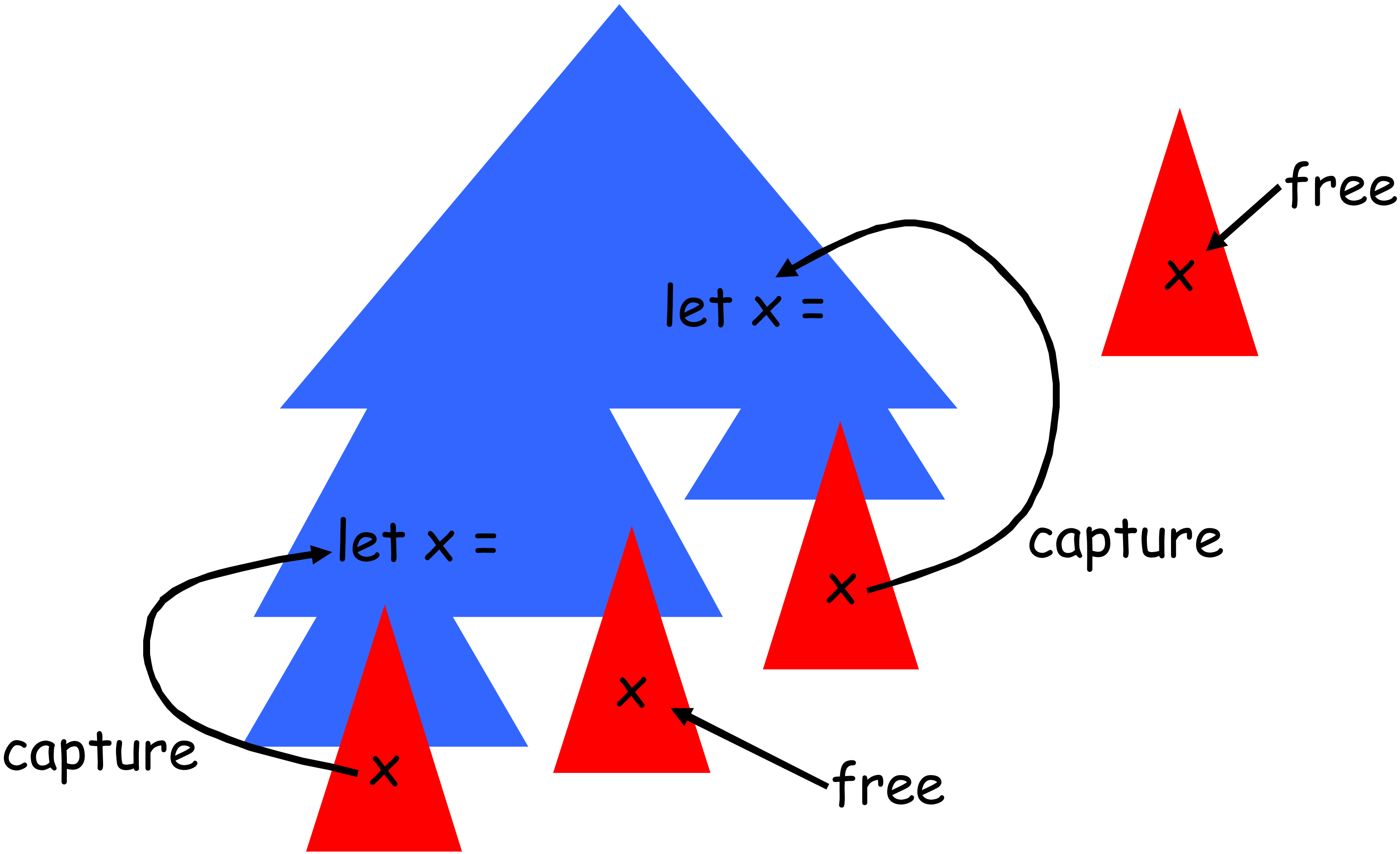
A Context



Filling the holes



Variable capture



Operational Equivalence

Two program phrases P and Q are operationally equivalent, $P \sim Q$

iff

for all contexts C[.] such that C[P] and C[Q] are complete programs,

the observable result of executing C[P] is the same as C[Q]

Also known as contextual equivalence or observational equivalence

Reasoning about $\sim$

Considered hard to reason from the definition because of the quantification $\forall C \ldots$

- Avoid this via alternative characterizations of $\sim$
- Bite the bullet…

Direct Reasoning with Contexts

- Not impossible
- We will use an applied lambda calculus as a running example.

Suppose we want to prove that

$$\forall C.\ C[M]\Downarrow \Leftrightarrow C[N]\Downarrow$$

- $P\Downarrow$ means that P terminates

Direct reasoning

- Want to reduce reasoning about C[M] and C[N] to reasoning about M and N directly.
- Suppose C[M]$\Downarrow$. We want to argue that C[N]$\Downarrow$ (and vice-versa).
- Proof idea: (induction on the length of the computation)

Direct Reasoning

Consider the first computation step

$$C[M] \mapsto M'$$

1. Either it depends on M
   - Examine whether a similar step is thus possible for N
2. Or it is independent of M and so C[N] can form a similar computation step

Parametric computation

- Reasoning about case 1. is specific to the property at hand.
- Reasoning about case 2. is essentially the same in all cases, but tricky to formalise.

Example: Fixed-point properties

- Suppose we have recursively defined constants

$$f \triangleq C_f[f]$$

- Computation rule

$$f \mapsto C_f[f]$$

Recursive constants

- We wish to prove that the behaviour of a recursive constant f is completely characterised by it's finite "unwindings"
- Observe termination. Operational approximation:
- $M \sqsubseteq N \Leftrightarrow \forall C.\ C[M]\Downarrow \Rightarrow C[N]\Downarrow$

The Unwinding Lemma

$$\forall n.\ C[f^n] \sqsubseteq M$$

$$\Leftrightarrow$$

$$C[f] \sqsubseteq M$$

where

$$f^0 \triangleq f^0$$

$$f^{n+1} \triangleq C_f[f^n]$$

How to prove syntactic continuity

The hard part of the property

$$(\forall\ n.\ C[f^n] \sqsubseteq M) \Rightarrow C[f] \sqsubseteq M$$

can be proved by "direct" reasoning about contexts (c.f. [Smith, MFPS'92])

Proof outline

Assume $\forall n.\ C[f^n] \sqsubseteq M$.

Take an arbitrary closing context D such that $D[C[f]]\Downarrow$.

We need to show that $D[M]\Downarrow$

Sufficient to show that if $D[C[f]]$ converges in m steps then $D[C[f^m]]$ converges.

Core of the Proof

- Examine the first computation step of D[C[f]]. Two cases

1. either it unwinds f, in which case we can argue that f^m can be unwound similarly, or
2. the computation step does not depend on f, and so the step is "parametric" in the hole

Computing with contexts

Goal:

- Make "case 2" reasoning precise by lifting operational semantics to contexts

$$C \mapsto D$$

- Compatible with hole filling

$$C[M] \mapsto D[M] \text{ (roughly)}$$

Applicability

- Types of semantics:
  - SOS rules, reduction context semantics, abstract machines, rule formats
- Types of property
  - Context lemmas
  - Fixed-point principles
  - Time & space semantics, unbounded nondeterminism

Hole filling does not commute with alpha-conversion

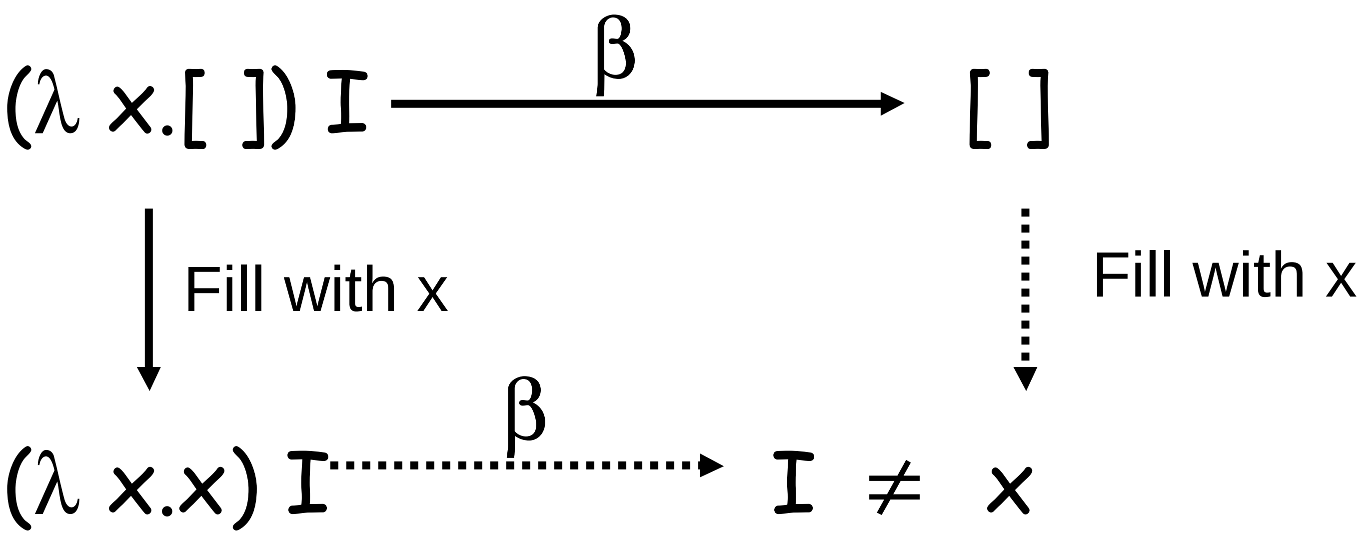
$(\lambda x.[\ ])\ \lambda x.\lambda y.[\ ] \longrightarrow (\lambda x.x\ y)\ \lambda x.\lambda y.x\ y$

α convert $\big\downarrow$

$(\lambda z.[\ ])\ \lambda x.\lambda y.[\ ] \longrightarrow (\lambda z.x\ y)\ \lambda x.\lambda y.x\ y$

Computation not compatible with hole filling

- If we treat holes as distinguished variables, we can compute:



Decorated Holes

- During computation, substitution applied to holes must be remembered
- $(l x.[\ ])\ I \rightarrow_{b} [\ ]^{\{x:=I\}}$

Once this extension has been admitted then we must allow nesting:

$(l y.[\ ])\ [\ ]^{\{x:=I\}} \rightarrow_{b} [\ ]^{\theta}$ where $\theta = [\ ]^{\{x:=I\}}$

The Talcott/Mason Approach

- Develop a calculus of contexts based on substitution-decorated holes
- Extend some specific computation rules to contexts
- Prove that context reduction is compatible with hole-filling
- Use this to prove operational equivalences

A Simpler Approach

1. Representing contexts in any language with variable binding using higher-order abstract syntax. No new calculus needed.
2. Represent definitions over terms (e.g. operational semantics rules) as HO syntax. Not specific to reduction relations
3. Automatically lift definitions to contexts; compatible with hole-filling "for free"

Computing with Contexts, A simple approach, ENTCS 10 (1998)

A Representation of Contexts

- A. Pitts, Notes on Inductive & Coinductive Techniques in the Semantics of Functional Languages, BRICS NS-94-5
  - Motivation: identify contexts up to alpha-equivalence
  - Related: Klop's CRS, Church'

Holes as functions

- Holes representing missing terms will be represented by first-order function variables x, x' with types of the form

$$(\text{Term},\ldots,\text{Term}) \to \text{Term}$$

- Hole filling corresponds to replacing hole variables by abstractions of the corresponding type

Example

- Conventional context

$$(\lambda x.[\ ])I$$

can be represented by

$$(\lambda x.\mathtt{x}(x))\ I$$

- $\mathtt{x}$ is a metavariable of type

$$\text{Term} \to \text{Term}$$

Example

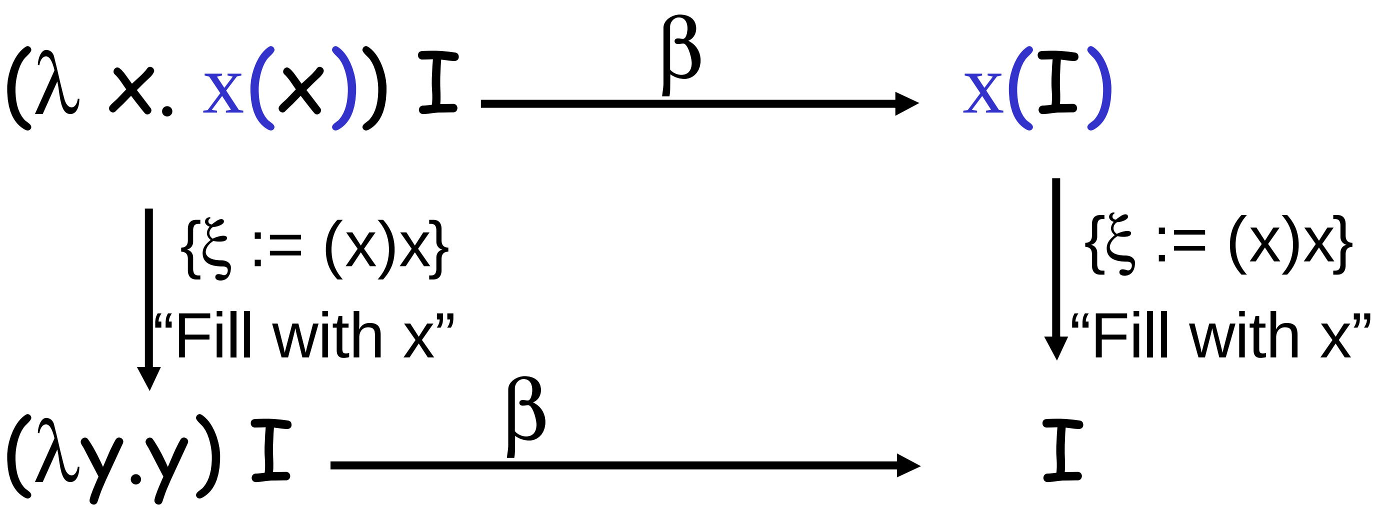
- Filling $(\lambda x.[\ ])I$ with term x

can be represented by substitution of the meta abstraction $(y)y$ for x

$$(\lambda x.x(x))\ I\ \{x := (x)x\}$$
$$= (\lambda y.x(y))\ I\ \{x := (z)z\} \quad (\alpha\text{-conv})$$
$$= (\lambda y.y)\ I$$

Example

- If we meta-applications as new constants we can compute with contexts:



Potential confusion

Entities of the form $x(x_1,\ldots,x_k)$ are meta-applications, not applications in the source language of our examples!

(Entities of the form $(x_1,\ldots,x_k)M$ are the corresponding meta-abstractions)

Hole variables

- Since we will only use metavariables of type $(Term_1,\dots,Term_k) \rightarrow Term$ (for some $k \geq 0$)
- Sufficient to refer to the arity of the hole metavariables
- $arity(\xi) = k$ means that ξ is an abstraction of type

$$(Term_1,\dots,Term_k) \rightarrow Term$$

Contexts

Contexts over a given language T, denoted T*, defined inductively as

- $C \in T^*$ whenever $C \in T$
- $\xi(C_1,\ldots,C_k) \in T^*$ whenever $\forall i \in 1\ldots k.\ C_i \in T^*$ and $\text{arity}(\xi) = k$

Hole filling

- Hole filling is defined by capture-avoiding substitution (i.e., the normal kind!)
- The only interesting case is

$x(C_1,\ldots,C_k)\theta$ where $\theta = \{\xi := (x_1,\ldots,x_n)D\}$

$= D\{x_1 := C_1\theta, \ldots, x_n := C_n\theta\}$

Hole filling

$x(C_1,\ldots,C_k)\theta$ where $\theta = \{\xi := (x_1,\ldots,x_n)D\}$
$= (x_1,\ldots,x_n)D \cdot (C_1\theta,\ldots,C_k\theta)$
$= D\{x_1 := C_1\theta,\ \ldots,\ x_n := C_n\theta\}$

We hide the beta reduction of this meta-term in the definition of substitution

Conventional Contexts

- Conventional contexts correspond to a special class of contexts, namely those with all holes of the form

$$x(x_1,\ldots,x_k) \text{ for some } x$$

- Contexts are identified up to renaming of bound variables

Representing Conventional Contexts

The representation of C is given by

$$\langle\, x \,\rangle = x$$

$$\langle\, [\;] \,\rangle = \mathrm{x}(z_1,\ldots z_n)$$

$$\langle\, op(C_1,\ldots,C_k) \,\rangle = op(\langle\, C_1 \,\rangle,\ldots,\langle\, C_k \,\rangle\,)$$

where $z_1,\ldots,z_n$ is a vector of all variables in scope at the holes in C

Exercise

- How can the context

$$(\lambda x.[\ ])\ ((\lambda x.[\ ])\ I)$$

be represented?

- Perform two beta-reductions on your context and confirm that these reductions "commute" with what you get by filling the hole with x.

Checkpoint

- Seen a functional representation of contexts (following *A.* Pitts notation)
- Examples suggest that the obvious notion of computation compatible with hole-filling

- To do: why it works - a general argument

Higher-order Abstract Syntax

- To generalise over syntax and syntactic definitions we use a higher-order abstract syntax

(widely used in type-theory, logical frameworks…)

Example

Concrete syntax $(\lambda x.y)\ z$

represented by

apply((lambda ((x)y)), z)

apply has type (term,term) $\rightarrow$ term

lambda has type (term $\rightarrow$ term) $\rightarrow$ term

Example

Concrete syntax $(\lambda x.y)\ z$

represented by

apply((lambda ((x)y)), z)

apply has arity (0,0)

lambda has arity (1)

Example

```
case M of
   nil => N;
   cons x xs => N'
```

Computation rules

Seen how higher-order abstract syntax can represent

- contexts and
- syntax involving variable binding

Now we look at how rules and inductive definitions can be represented

Computation rules

Computation rules, e.g.

$$(\lambda x.M)\ N \mapsto M\{x := N\}$$

$$\frac{M \mapsto M'}{M\ N \mapsto M'\ N}$$

represented using typed metavariables

X, Y, Z

Formal Computation Rules

$$\text{apply}(\text{lambda } X, Y) \mapsto X\, Y$$

$$\frac{Y \mapsto Y'}{\text{apply}(Z,Y) \mapsto \text{apply}(Z,Y')}$$

Instance of a rule obtained by mapping metavariables to abstractions (and normalising)

Formal Computation Rules

Example, $\{X := (z)z, Y := 3\}$

- applied to

  $$\text{apply}(\text{lambda } X, Y) \mapsto X\ Y$$

- gives instance

  $$\text{apply}(\text{lambda } (z)z, 3) \mapsto 3$$

Computing with Contexts

Simply allow instances of rules to contain holes!

apply(lambda X, Y) $\mapsto$ X Y

when X := (z)x(z), Y := x(y)

yields

apply(lambda (z)x(z), x(y)) $\mapsto$ x(x(y))

Why it works

- A rule (e.g. an axiom like the beta reduction rule) is a pair of meta-terms

$$\textbf{Rule} = \langle L, R \rangle$$

terms containing metavariables

Why it works

- A rule (e.g. an axiom like the beta reduction rule) is a pair of meta-terms

Rule = $\langle L, R \rangle$

Instance = $\langle L\sigma, R\sigma \rangle$

instance containing holes

Why it works

- A rule (e.g. an axiom like the beta reduction rule) is a pair of meta-terms

$$\textbf{Rule} = \langle L, R \rangle$$

$$\textbf{Instance} = \langle L\sigma, R\sigma \rangle$$

$$\textbf{Hole filling} = \langle (L\sigma)\tau, (R\sigma)\tau \rangle$$

filling the holes

Why it works

Is $\langle (L\sigma)\tau, (R\sigma)\tau \rangle$ a valid instance?

i.e. if we compute with contexts then fill the holes, is that the same as filling the holes and computing? Since metavariables and hole variables are distinct

$$\langle (L\sigma)\tau, (R\sigma)\tau \rangle = \langle L(\sigma\tau), R(\sigma\tau) \rangle$$

I.e. the substitution lemma from lambda-calculus

Applicability

- The argument generalises to any inductively defined relation
- Configurations of SOS rules
- Particular subsets of terms
- $V \in Values ::= \langle V_1, V_2 \rangle \mid \lambda x.M$
- Evaluation contexts

Applicability

- GDSOS rule format [Sands POPL'97]
  - various theorems that hold for any functional language fitting the format
- Context lemmas for call-by-need [Moran & Sands POPL'99]
- Theory of Space improvement [Gustavsson & Sands, ICFP'2001]

Conclusion

A simple typed-lambda-calculus representation of contexts

- lifts definitions to contexts
- compatible with hole filling
- useful for reasoning about operational equivalence

Related Work

The “direct reasoning” style

- Talcott, Mason, Smith, Felleisen

e.g. Mason and Talcottm Equivalence in functional languages with effects [JFP 1991]

Related Work

Typed lambda calculus representation of contexts

- Huet and Lang, Proving and Applying program transformations expressed with second-order patters [Acta Inf '78]
- Klop, PhD thesis
- Pitts tutorial BRICS 1994

Related Work

- Context calculi
  - Talcott
  - Mason
  - Hashimoto & Ohori
  - Lee & Freedman

May provide more generality in some cases

Further Work

- Test the applicability in nominal calculi
  - Potential pitfalls(?): clauses depending on the equivalence or inequivalence of names

- www.cs.chalmers.se/~dave/SOS04/