Nondeterminism, fixpoints and bisimulation

Paul Blain Levy

University of Birmingham

November 12, 2010

Outline

1. Setting Up
   - Imperative and functional languages
   - Adding nondeterminism
2. Linear time equivalences
   - May testing: imperative and functional
   - Infinite traces: imperative
   - Seeing Beyond Divergence: imperative
   - May and must testing: functional
3. Branching time equivalence: imperative and functional
4. Lines of attack

Functional language: call-by-name FPC

Types

$$A ::= \quad A \to A \mid \sum_{i \in I} A_i \mid \prod_{i \in I} A_i \mid \mathtt{X} \mid \mathtt{rec}\ \mathtt{X}.\ A \quad (I \text{ countable})$$

Terms

$$\begin{array}{rcl} M & ::= & \mathtt{x} \mid \langle i, M \rangle \mid \mathtt{match}\ M\ \mathtt{as}\ \{\langle i, \mathtt{x} \rangle . M_i\}_{i \in I} \\ & & \mid \lambda \mathtt{x}.M \mid MM \mid Mi \mid \lambda\{i.M_i\}_{i \in I} \\ & & \mid \mathtt{rec}\ \mathtt{x}.\ M \mid \mathtt{fold}\ M \mid \mathtt{unfold}\ M \end{array}$$

Functional language: call-by-name FPC

Types

$$A ::= \quad A \to A \mid \sum_{i \in I} A_i \mid \prod_{i \in I} A_i \mid \mathtt{X} \mid \mathtt{rec}\ \mathtt{X}.\ A \quad (I \text{ countable})$$

Terms

$$\begin{array}{rcl} M & ::= & \mathtt{x} \mid \langle i, M \rangle \mid \mathtt{match}\ M\ \mathtt{as}\ \{\langle i, \mathtt{x} \rangle.M_i\}_{i \in I} \\ & & \mid \lambda \mathtt{x}.M \mid MM \mid Mi \mid \lambda\{i.M_i\}_{i \in I} \\ & & \mid \mathtt{rec}\ \mathtt{x}.\ M \mid \mathtt{fold}\ M \mid \mathtt{unfold}\ M \end{array}$$

A ground type is $\sum_{i \in I} 1$

The strategy types are of the form $\prod \sum \prod \sum \prod \sum \ldots$

Cpo semantics: $\sum$ denotes lifted sum

Big-step semantics (Cousot)

Convergence

Terminal terms $T ::= \quad \lambda \mathrm{x}.M \mid \lambda\{i.M_i\}_{i \in I} \mid \langle i, M \rangle$

Define convergence $M \Downarrow T$ inductively, e.g.

$$\frac{M \Downarrow \lambda \mathrm{x}.P \quad P[N/\mathrm{x}] \Downarrow T}{MN \Downarrow T}$$

Big-step semantics (Cousot)

Convergence

Terminal terms $T ::= \quad \lambda \mathrm{x}.M \mid \lambda\{i.M_i\}_{i \in I} \mid \langle i, M \rangle$

Define convergence $M \Downarrow T$ inductively, e.g.

$$\frac{M \Downarrow \lambda \mathrm{x}.P \qquad P[N/\mathrm{x}] \Downarrow T}{MN \Downarrow T}$$

Divergence

Define divergence $M \Uparrow$ coinductively, e.g.

$$\frac{M \Downarrow \lambda \mathrm{x}.P \qquad P[N/\mathrm{x}] \Uparrow}{MN \Uparrow}$$

Imperative Language

Syntax

$$M ::= \quad \texttt{print}\ c.\ M \mid \texttt{x} \mid \texttt{rec x.}\ M \qquad\qquad c \in \mathcal{A}$$

Also allow countable mutual recursion.

Imperative Language

Syntax

$$M ::= \quad \texttt{print}\ c.\ M \mid \texttt{x} \mid \texttt{rec x.}\ M \qquad\qquad c \in \mathcal{A}$$

Also allow countable mutual recursion.

Small-step semantics

$$\begin{array}{rcl} \texttt{print}\ c.\ M & \overset{c}{\rightsquigarrow} & M \\ \texttt{rec x.}\ M & \rightsquigarrow & M[\texttt{rec x.}\ M/\texttt{x}] \end{array}$$

Imperative Language

Syntax

$$M ::= \quad \texttt{print}\ c.\ M \mid \texttt{x} \mid \texttt{rec x.}\ M \qquad\qquad c \in \mathcal{A}$$

Also allow countable mutual recursion.

Small-step semantics

$$\begin{array}{rcl} \texttt{print}\ c.\ M & \overset{c}{\rightsquigarrow} & M \\ \texttt{rec x.}\ M & \rightsquigarrow & M[\texttt{rec x.}\ M/\texttt{x}] \end{array}$$

A program either

- prints a finite string, then diverges
- or prints an infinite string.

Medium step semantics

Convergence

Define $M \overset{c}{\Rightarrow} N$ inductively:

$$\frac{}{\texttt{print}\ c.\ M \overset{c}{\Rightarrow} M} \qquad \frac{M[\texttt{rec x}.\ M/\texttt{x}] \overset{c}{\Rightarrow} N}{\texttt{rec x}.\ M \overset{c}{\Rightarrow} N}$$

Divergence

Define $M \Uparrow$ coinductively:

$$\frac{M[\texttt{rec x}.\ M/\texttt{x}] \Uparrow}{\texttt{rec x}.\ M \Uparrow}$$

Medium step semantics

Convergence

Define $M \overset{c}{\Rightarrow} N$ inductively:

$$\frac{}{\texttt{print } c.\ M \overset{c}{\Rightarrow} M} \qquad \frac{M[\texttt{rec x. } M/\texttt{x}] \overset{c}{\Rightarrow} N}{\texttt{rec x. } M \overset{c}{\Rightarrow} N}$$

Divergence

Define $M \Uparrow$ coinductively:

$$\frac{M[\texttt{rec x. } M/\texttt{x}] \Uparrow}{\texttt{rec x. } M \Uparrow}$$

We have

- $M \overset{c}{\Rightarrow} N$ iff $M \rightsquigarrow^* \overset{c}{\rightsquigarrow} N$
- $M \Uparrow$ iff $M \rightsquigarrow^\omega$

From imperative to functional

Define the strategy type $$\mathtt{Proc} \stackrel{\text{def}}{=} \sum_{c \in \mathcal{A}} \mathtt{Proc}$$

$\mathtt{x}_0, \ldots, \mathtt{x}_{n-1} \vdash M$ in the imperative language

translates into $\mathtt{x}_0 : \mathtt{Proc}, \ldots, \mathtt{x}_{n-1} : \mathtt{Proc} \vdash M : \mathtt{Proc}$ in the functional language.

From imperative to functional

Define the strategy type $\quad \texttt{Proc} \stackrel{\mathrm{def}}{=} \sum_{c \in \mathcal{A}} \texttt{Proc}$

$\mathtt{x}_0, \ldots, \mathtt{x}_{n-1} \vdash M$ in the imperative language

translates into $\mathtt{x}_0 : \texttt{Proc}, \ldots, \mathtt{x}_{n-1} : \texttt{Proc} \vdash M : \texttt{Proc}$ in the functional language.

This translation preserves operational semantics.

From imperative to functional

Define the strategy type $\quad \texttt{Proc} \stackrel{\mathrm{def}}{=} \sum_{c \in \mathcal{A}} \texttt{Proc}$

$\texttt{x}_0, \ldots, \texttt{x}_{n-1} \vdash M$ in the imperative language

translates into $\texttt{x}_0 : \texttt{Proc}, \ldots, \texttt{x}_{n-1} : \texttt{Proc} \vdash M : \texttt{Proc}$ in the functional language.

This translation preserves operational semantics.

Since Proc denotes the domain $\mathcal{A}^{*\omega}$, it preserves cpo semantics too.

From imperative to functional

Define the strategy type $\qquad \mathtt{Proc} \stackrel{\mathrm{def}}{=} \sum_{c \in \mathcal{A}} \mathtt{Proc}$

$\mathtt{x}_0, \ldots, \mathtt{x}_{n-1} \vdash M$ in the imperative language

translates into $\mathtt{x}_0 : \mathtt{Proc}, \ldots, \mathtt{x}_{n-1} : \mathtt{Proc} \vdash M : \mathtt{Proc}$ in the functional language.

This translation preserves operational semantics.

Since Proc denotes the domain $\mathcal{A}^{*\omega}$, it preserves cpo semantics too.

We could also translate interactive input, using nontrivial $\prod$.

Three kinds of nondeterminism

We can add to the functional language various kinds of nondeterminism.

Binary erratic nondeterminism M `or` M'

Choose to go left (and evaluate M) or right (and evaluate M')

Countable erratic nondeterminism `choose` $n \in \mathbb{N}.\ M_n$

Choose a number n, then evaluate M_n

Ambiguous nondeterminism M `amb` M'

Evaluate M and M' fairly, return whatever you get first.
If M returns after 1 step, and M' returns after 10000 steps, could still return the latter.
We require M, M' to have $\sum$ type.

Ground amb

Provides `amb` at ground type only.

Expressiveness

Define

$$\begin{aligned} \perp &\stackrel{\mathrm{def}}{=} \mathtt{rec\ x.\ x} \\ \mathtt{choose}^{\perp}\ n \in \mathbb{N}.\ M_n &\stackrel{\mathrm{def}}{=} \perp \mathtt{\ or\ choose}\ n \in \mathbb{N}.\ M_n \end{aligned}$$

- Binary erratic nondeterminism can express $\mathtt{choose}^{\perp}\ n \in \mathbb{N}.\ M_n$.

$$\begin{tikzcd} \bullet \arrow[r] \arrow[dr] & \bullet \arrow[r] \arrow[dr] & \bullet \arrow[r] \arrow[dr] & \bullet \arrow[r] \arrow[dr] & \cdots \\ & M_0 & M_1 & M_2 & M_3 \end{tikzcd}$$

- Ground `amb` can express countable erratic nondeterminism, parallel-or and parallel-exists.

Example Application

Example Application

The program must not kill the customer.

Example Application

The program must not kill the customer.

safety property

Example Application

The program must not kill the customer.
safety property

The program must greet the customer.

Example Application

The program must not kill the customer.
safety property

The program must greet the customer.
liveness property

Example Application

The program must not kill the customer.
safety property

The program must greet the customer.
liveness property

If the program insults the customer, it must apologize.

Example Application

The program must not kill the customer.
safety property

The program must greet the customer.
liveness property

If the program insults the customer, it must apologize.
conditional liveness property

Example Application

The program must not kill the customer.
safety property

The program must greet the customer.
liveness property

If the program insults the customer, it must apologize.
conditional liveness property

The program must stop insulting the customer.

Example Application

The program must not kill the customer.
safety property

The program must greet the customer.
liveness property

If the program insults the customer, it must apologize.
conditional liveness property

The program must stop insulting the customer.
infinite liveness property

May testing

The most basic equivalence on programs is may testing.

This asks: what are the things that we may observe?

Or equivalently, the things that we definitely won't observe (safety properties).

May testing

The most basic equivalence on programs is may testing.
This asks: what are the things that we may observe?
Or equivalently, the things that we definitely won't observe (safety properties).

Definition

$M \simeq_{\mathsf{may}} M'$ when, for every ground context $\mathcal{C}[\cdot]$,

$$\mathcal{C}[M] \Downarrow n \Leftrightarrow \mathcal{C}[M'] \Downarrow n$$

Examples

$$\begin{array}{rcl} M \ \mathtt{or}\ \bot & \simeq_{\mathsf{may}} & M \\ M \ \mathtt{amb}\ M' & \simeq_{\mathsf{may}} & M \ \mathtt{or}\ M' \end{array}$$

In the imperative language, closed terms M and M' are identified when they have the same finite traces.

Rational continuity

May testing has a continuity property that gives rise to lower powerdomain semantics.

Definition of $\texttt{rec}^n\ \texttt{x}.\ M$

$$\begin{array}{rcl} \texttt{rec}^0\ \texttt{x}.\ M & \stackrel{\text{def}}{=} & \bot \\ \texttt{rec}^{n+1}\ \texttt{x}.\ M & \stackrel{\text{def}}{=} & M[\texttt{rec}^n\ \texttt{x}.\ M/\texttt{x}] \end{array}$$

Theorem

If $\mathcal{C}[\texttt{rec}\ \texttt{x}.\ M]$ can print "hello", then there exists $n \in \mathbb{N}$ such that $\mathcal{C}[\texttt{rec}^n\ \texttt{x}.\ M]$ can print "hello".

Cpo semantics for may testing is typically fully definable and fully abstract.

Translation doesn't preserve may-testing

Imperative	$a.(b.\bot \texttt{ or } c.\bot)$	$\simeq_{\mathsf{may}}$	$a.b.\bot \texttt{ or } a.c.\bot$
Functional	$\langle a, \langle b, \bot\rangle \texttt{ or } \langle c, \bot\rangle\rangle$		$\langle a, \langle b, \bot\rangle\rangle \texttt{ or } \langle a, \langle c, \bot\rangle\rangle$

These can be distinguished by the context

$$\texttt{match } [\cdot] \texttt{ as} \begin{cases} \langle a, \mathtt{x}\rangle. & \texttt{match x as} \begin{cases} \langle b, \mathtt{y}\rangle. & \texttt{match x as} \begin{cases} \langle c, \mathtt{z}\rangle. & \texttt{true} \\ \langle \neq c, \mathtt{z}\rangle. & \bot \end{cases} \\ \langle \neq b, \mathtt{y}\rangle. & \bot \end{cases} \\ \langle \neq a, \mathtt{x}\rangle. & \bot \end{cases}$$

To rectify this, we need an affine target language (like Winskel's Affine HOPLA).

Imperative language: infinite traces

For a closed term M, its set of behaviours $[M] \in \mathcal{P}(\mathcal{A}^{*\infty})$

Imperative language: infinite traces

For a closed term M, its set of behaviours $[M] \in \mathcal{P}(\mathcal{A}^{*\infty})$

The kernel of $[\,]$ is called infinite trace equivalence.

Imperative language: infinite traces

For a closed term M, its set of behaviours $[M] \in \mathcal{P}(\mathcal{A}^{*\infty})$

The kernel of $[\,]$ is called infinite trace equivalence.

Probably the most obvious equivalence to consider.

Imperative language: infinite traces

For a closed term M, its set of behaviours $[M] \in \mathcal{P}(\mathcal{A}^{*\infty})$

The kernel of $[\,]$ is called infinite trace equivalence.

Probably the most obvious equivalence to consider.

Can recognize all the properties of our customer service program.

Imperative language: infinite traces

For a closed term M, its set of behaviours $[M] \in \mathcal{P}(\mathcal{A}^{*\infty})$

The kernel of $[\,]$ is called infinite trace equivalence.

Probably the most obvious equivalence to consider.

Can recognize all the properties of our customer service program.

Can we give a denotational semantics that agrees with $[\,]$ on closed terms?

Infinite traces
What doesn't work (1): least fixpoint semantics

In least fixpoint semantics, $\bot$ is the least fixpoint of the identity, so $\bot \leqslant M$.

Consider

$$\begin{array}{rcl} M & \stackrel{\text{def}}{=} & \bot \ \texttt{or} \ \text{insult.apol.}\bot \\ M' & \stackrel{\text{def}}{=} & \bot \ \texttt{or} \ \text{insult.}\bot \ \texttt{or} \ \text{insult.apol.}\bot \end{array}$$

We have

$$\begin{array}{rcl} M = & \bot \ \texttt{or} \ \bot \ \texttt{or} \ \text{insult.apol.}\bot & \leqslant M' \\ M = & \bot \ \texttt{or} \ \text{insult.apol.}\bot \ \texttt{or} \ \text{insult.apol.}\bot & \geqslant M' \end{array}$$

So $M = M'$, contradicting infinite trace equivalence.

Infinite traces
What doesn't work (2): well-pointed semantics

In well-pointed semantics, a term in context Γ denotes a function from a set of environments.

Linked to a context lemma: two terms that are equivalent in every environment are equivalent in every program context.

That is false for our language, in the case that $\mathcal{A} = \{\checkmark\}$.

Infinite traces
Counterexample to context lemma

Here are two terms with a free identifier x.

$$
\begin{array}{rcl}
N & = & \texttt{choose}^{\perp}\ n \in \mathbb{N}.\ \checkmark^{n}.\ \perp\ \texttt{or x} \\
N' & = & \texttt{choose}^{\perp}\ n \in \mathbb{N}.\ \checkmark^{n}.\ \perp\ \texttt{or x or}\ \checkmark.\texttt{x}
\end{array}
$$

	$\checkmark^{n}$, then diverge	$\checkmark^{\omega}$
N	yes	iff x can
N'	yes	iff x can
rec x. N	yes	no
rec x. N'	yes	yes

Infinite traces
What does work: intensional semantics

$$
\begin{array}{rcl}
N & = & \texttt{choose}^{\perp}\ n \in \mathbb{N}.\ \checkmark^{n}.\ \perp\ \texttt{or}\ \texttt{x} \\
N' & = & \texttt{choose}^{\perp}\ n \in \mathbb{N}.\ \checkmark^{n}.\ \perp\ \texttt{or}\ \texttt{x}\ \texttt{or}\ \checkmark.\texttt{x}
\end{array}
$$

Somehow we have to distinguish N and N'.

What does work: intensional semantics

$$
\begin{aligned}
N &= \texttt{choose}^{\perp}\ n \in \mathbb{N}.\ \checkmark^{n}.\ \perp\ \texttt{or}\ \texttt{x} \\
N' &= \texttt{choose}^{\perp}\ n \in \mathbb{N}.\ \checkmark^{n}.\ \perp\ \texttt{or}\ \texttt{x}\ \texttt{or}\ \checkmark.\texttt{x}
\end{aligned}
$$

Somehow we have to distinguish N and N'.

Game semantics

N' can print $\checkmark$, then force (i.e. execute) $\texttt{x}$.
Make forcing explicit in the denotational semantics.

Infinite traces
What does work: intensional semantics

$$\begin{array}{rcl} N & = & \mathtt{choose}^{\perp}\ n \in \mathbb{N}.\ \checkmark^{n}.\ \perp\ \mathtt{or}\ \mathtt{x} \\ N' & = & \mathtt{choose}^{\perp}\ n \in \mathbb{N}.\ \checkmark^{n}.\ \perp\ \mathtt{or}\ \mathtt{x}\ \mathtt{or}\ \checkmark.\mathtt{x} \end{array}$$

Somehow we have to distinguish N and N'.

Game semantics

N' can print $\checkmark$, then force (i.e. execute) $\mathtt{x}$.
Make forcing explicit in the denotational semantics.

Presheaf semantics

Interpret N and N' using alphabet $\mathcal{A} + 1$.
For n free identifiers, use alphabet $\mathcal{A} + n$.

Seeing Beyond Divergence (1)

Definition of $[M]_{\mathfrak{I}}$

- the set of finite traces of M, together with extensions of divergences
- the set of extensions of divergences of M
- the set of infinite traces of M, together with extensions of divergences

This semantics is divergence strict.

Seeing Beyond Divergence (1)

Definition of $[M]_{\mathfrak{I}}$

- the set of finite traces of M, together with extensions of divergences
- the set of extensions of divergences of M
- the set of infinite traces of M, together with extensions of divergences

This semantics is divergence strict.

To model recursion, we take the greatest fixpoint. (Reverse ordering is the upper powerdomain.)

Seeing Beyond Divergence (2)

Definition of $[M]_{\mathsf{SBD}}$

- the set of finite traces of M
- the set of divergences of M
- the set of infinite traces, together with limits of divergences (called "ω-divergences")

Seeing Beyond Divergence (2)

Definition of $[M]_{\mathsf{SBD}}$

- the set of finite traces of M
- the set of divergences of M
- the set of infinite traces, together with limits of divergences (called "ω-divergences")

To model recursion:

- first compute the greatest fixpoint wrt $[\,]_{\mathfrak{I}}$, giving an equivalence class for $[\,]_{\mathsf{SBD}}$
- then compute the least fixpoint wrt $[\,]_{\mathsf{SBD}}$ within that class.

This is called the reflected fixpoint.

Functional language: may and must testing

For a set $A \subseteq \mathbb{N}$, we want to observe whether a program must return a value in A.

This is a liveness property.

Definition

For two terms M, M', say $M \simeq_{\mathsf{may-must}} M'$ when for every ground context $\mathcal{C}[\cdot]$, we have

$$\begin{array}{rcl} \mathcal{C}[M] \Downarrow n & \Leftrightarrow & \mathcal{C}[M'] \Downarrow n \\ \mathcal{C}[M] \Uparrow & \Leftrightarrow & \mathcal{C}[M'] \Uparrow \end{array}$$

The context lemma holds for (binary or countable) erratic nondeterminism under this equivalence. [Lassen]

Powerdomain semantics for may-must: binary nondeterminism

For binary nondeterminism, we have a continuity property for must-testing.

Theorem

For any $A \subseteq \mathbb{N}$, if $\mathcal{C}[\texttt{rec x.}\ M]$ must return an element of A, then there exists n such that $\mathcal{C}[\texttt{rec}^n\ \texttt{x.}\ M]$ must return an element of A.

This leads to convex powerdomain semantics for $\simeq_{\mathsf{may-must}}$ [Plotkin].

Powerdomain semantics for may-must: binary nondeterminism

For binary nondeterminism, we have a continuity property for must-testing.

Theorem

For any $A \subseteq \mathbb{N}$, if $\mathcal{C}[\mathtt{rec}\ \mathtt{x}.\ M]$ must return an element of A, then there exists n such that $\mathcal{C}[\mathtt{rec}^n\ \mathtt{x}.\ M]$ must return an element of A.

This leads to convex powerdomain semantics for $\simeq_{\mathsf{may-must}}$ [Plotkin].

Problem The model contains undefinable elements even at first order, causing failure of full abstraction at second order.

May-must equivalence for countable nondeterminism
What doesn't work: continuity

Continuous semantics cannot recognize divergence.

Proof (Apt-Plotkin)

Define $A \stackrel{\text{def}}{=} \prod_{n \in \mathbb{N}} \mathtt{bool}$ and define $\mathtt{f} : A \vdash M : A$ to be

$$\lambda \begin{cases} 0. & \mathtt{choose}\ n > 0.\ \mathtt{f}(n) \\ 1. & \mathtt{true} \\ n > 1. & \mathtt{f}(n-1) \end{cases}$$

and $\mathcal{C}[\cdot]$ to be $[\cdot]0$. Then, up to may-must equivalence,

$$\begin{array}{rcl} \mathcal{C}[\mathtt{rec}^k\ \mathtt{f}.\ M] & \text{is} & \mathtt{true}\ \mathtt{or}\ \bot \\ \mathcal{C}[\mathtt{rec}\ \mathtt{f}.\ M] & \text{is} & \mathtt{true} \end{array}$$

Plotkin et al developed a variant of the convex powerdomain for countable nondeterminism, using transfinite approximants.

Amb
What doesn't work (1): least fixpoint semantics

How can we give denotational semantics for amb?

Least fixpoint semantics doesn't work, even for ground amb.

$$\texttt{true or } \bot \leqslant \texttt{true or true} = \texttt{true}$$

$$\begin{array}{rcl} \texttt{true} & = & \texttt{if } (\texttt{false amb } \bot) \texttt{ then } \bot \texttt{ else true} \\ & \leqslant & \texttt{if } (\texttt{false amb true}) \texttt{ then } \bot \texttt{ else true} \\ & = & \texttt{true or } \bot \end{array}$$

So $\texttt{true or } \bot = \texttt{true}$. That's may-testing.

Amb
What doesn't work (2): well-pointed semantics

[MFPS 2007] Amb breaks the context lemma.

Let A be the strategy type $\prod_1 \sum_1 \prod_1 \sum_1 1$.

A closed term of type A gives (operationally) an element of
$[A] \stackrel{\mathrm{def}}{=} \mathcal{P}((\mathcal{P}(1_\perp)_\perp)$.

Amb
What doesn't work (2): well-pointed semantics

[MFPS 2007] Amb breaks the context lemma.

Let A be the strategy type $\prod_1 \sum_1 \prod_1 \sum_1 1$.

A closed term of type A gives (operationally) an element of $[A] \stackrel{\text{def}}{=} \mathcal{P}((\mathcal{P}(1_\perp)_\perp)$.

There exist two terms $\mathtt{x} : A \vdash M, M' : A$ giving (operationally) the same endofunction on $[A]$

and a ground context $\mathcal{C}[\cdot]$ such that

- $\mathcal{C}[\mathtt{rec\ x.}\ M]$ may diverge
- $\mathcal{C}[\mathtt{rec\ x.}\ M']$ must converge.

So no well-pointed semantics is possible.

Context lemma for ground amb

[MFPS 2007] The context lemma holds in the presence of ground amb.

This suggests that there could be a well-pointed semantics for ground amb.

Lower and convex bisimulation: imperative language

Let $\mathcal{R}$ be a binary relation on closed terms.

It is a lower simulation when $M \mathrel{\mathcal{R}} M'$ and $M \overset{c}{\Rightarrow} N$ implies $\exists N'$ such that $M' \overset{c}{\Rightarrow} N'$ and $N \mathrel{\mathcal{R}} N'$.

It is a lower bisimulation when $\mathcal{R}$ and $\mathcal{R}^{\mathrm{op}}$ are lower simulations.

It is a convex bisimulation when moreover $M \mathrel{\mathcal{R}} M'$ implies $M \Uparrow \Leftrightarrow M' \Uparrow$.

The greatest lower bisimulation is called lower bisimilarity.

Lower and convex bisimulation: imperative language

Let $\mathcal{R}$ be a binary relation on closed terms.

It is a lower simulation when $M \mathrel{\mathcal{R}} M'$ and $M \stackrel{c}{\Rightarrow} N$ implies $\exists N'$ such that $M' \stackrel{c}{\Rightarrow} N'$ and $N \mathrel{\mathcal{R}} N'$.

It is a lower bisimulation when $\mathcal{R}$ and $\mathcal{R}^{\mathrm{op}}$ are lower simulations.

It is a convex bisimulation when moreover $M \mathrel{\mathcal{R}} M'$ implies $M \Uparrow \Leftrightarrow M' \Uparrow$.

The greatest lower bisimulation is called lower bisimilarity.

Two terms are lower bisimilar

Lower and convex bisimulation: imperative language

Let $\mathcal{R}$ be a binary relation on closed terms.

It is a lower simulation when $M \mathrel{\mathcal{R}} M'$ and $M \stackrel{c}{\Rightarrow} N$ implies $\exists N'$ such that $M' \stackrel{c}{\Rightarrow} N'$ and $N \mathrel{\mathcal{R}} N'$.

It is a lower bisimulation when $\mathcal{R}$ and $\mathcal{R}^{\mathrm{op}}$ are lower simulations.

It is a convex bisimulation when moreover $M \mathrel{\mathcal{R}} M'$ implies $M \Uparrow \Leftrightarrow M' \Uparrow$.

The greatest lower bisimulation is called lower bisimilarity.

Two terms are lower bisimilar

- iff they satisfy the same formulas in Hennessy-Milner logic

Lower and convex bisimulation: imperative language

Let $\mathcal{R}$ be a binary relation on closed terms.

It is a lower simulation when $M \; \mathcal{R} \; M'$ and $M \stackrel{c}{\Rightarrow} N$ implies $\exists N'$ such that $M' \stackrel{c}{\Rightarrow} N'$ and $N \; \mathcal{R} \; N'$.

It is a lower bisimulation when $\mathcal{R}$ and $\mathcal{R}^{\mathrm{op}}$ are lower simulations.

It is a convex bisimulation when moreover $M \; \mathcal{R} \; M'$ implies $M \Uparrow \Leftrightarrow M' \Uparrow$.

The greatest lower bisimulation is called lower bisimilarity.

Two terms are lower bisimilar

- iff they satisfy the same formulas in Hennessy-Milner logic
- iff there is a strategy for the bisimilarity game between them

Lower and convex bisimulation: imperative language

Let $\mathcal{R}$ be a binary relation on closed terms.

It is a lower simulation when $M\ \mathcal{R}\ M'$ and $M \stackrel{c}{\Rightarrow} N$ implies $\exists N'$ such that $M' \stackrel{c}{\Rightarrow} N'$ and $N\ \mathcal{R}\ N'$.

It is a lower bisimulation when $\mathcal{R}$ and $\mathcal{R}^{\mathrm{op}}$ are lower simulations.

It is a convex bisimulation when moreover $M\ \mathcal{R}\ M'$ implies $M \Uparrow \Leftrightarrow M' \Uparrow$.

The greatest lower bisimulation is called lower bisimilarity.

Two terms are lower bisimilar

- iff they satisfy the same formulas in Hennessy-Milner logic
- iff there is a strategy for the bisimilarity game between them
- iff they have the same anamorphic image.

Bisimilarity
What doesn't work: least fixpoint semantics

Once again

$$\begin{array}{rcl} M & \stackrel{\text{def}}{=} & \bot \texttt{ or } \mathsf{insult.apol.}\bot \\ M' & \stackrel{\text{def}}{=} & \bot \texttt{ or } \mathsf{insult.}\bot \texttt{ or } \mathsf{insult.apol.}\bot \end{array}$$

We have

$$\begin{array}{rcr} M = & \bot \texttt{ or } \bot \texttt{ or } \mathsf{insult.apol.}\bot & \leqslant M' \\ M = & \bot \texttt{ or } \mathsf{insult.apol.}\bot \texttt{ or } \mathsf{insult.apol.}\bot & \geqslant M' \end{array}$$

So $M = M'$, but they are not lower bisimilar.

Lower similarity
What doesn't work: continuity [Boudol, Abramsky, Lassen]

Let the alphabet be $\mathbb{N}$, and include a renaming operator $[+1]$.

Let $\mathtt{x} \vdash M$ be $\bot$ $\mathtt{or}$ $(0.\ \bot)$ $\mathtt{or}$ $[+1]\mathtt{x}$

Let $\mathcal{C}[\cdot]$ be $(\mathtt{choose}^{\bot}\ n \in \mathbb{N}.\ 0.\ \mathtt{choose}^{\bot}\ m \leqslant n.\ m.\ \bot)$ $\mathtt{or}$ $(0.\ [\cdot])$

Then. up to convex bisimilarity,

$$\begin{array}{rcl} \mathcal{C}[\mathtt{rec}^k\ \mathtt{x}.\ M] & \text{is} & (\mathtt{choose}^{\bot}\ n \in \mathbb{N}.\ 0.\ \mathtt{choose}^{\bot}\ m \leqslant n.\ m.\ \bot) \\ \mathcal{C}[\mathtt{rec}\ \mathtt{x}.\ M] & \text{is} & (\mathtt{choose}^{\bot}\ n \in \mathbb{N}.\ 0.\ \mathtt{choose}^{\bot}\ m \leqslant n.\ m.\ \bot) \\ & & \mathtt{or}\ (0.\ \mathtt{choose}^{\bot}\ m \in \mathbb{N}.\ m.\ \bot) \end{array}$$

The latter and the former are not related by lower similarity.

But they are identified by any continuous semantics.

Denotational semantics of lower bisimilarity

Is there a well-pointed semantics of lower bisimilarity?

Abramsky's domain equation

Abramsky presented a "domain equation for bisimulation".

If M, M' have no divergences then $[\![M]\!] = [\![M']\!]$ iff M, M' are lower bisimilar.

But for general programs, that is not the case.

What kind of fixpoint should we use to interpret recursion?

Applicative bisimilarity [Abramsky]: functional language

A binary relation $\mathcal{R}$ on closed terms is a lower applicative simulation when $M \mathrel{\mathcal{R}} M' : A$ implies

- (if $A = B \to C$) for all closed $N : B$ we have $MN \mathrel{\mathcal{R}} M'N$
- (if $A = \prod_{i \in I} B_i$) for all $i \in I$ we have $Mi \mathrel{\mathcal{R}} M'i$
- (if $A = \sum_{i \in I} A_i$) if $M \Downarrow \langle i, N \rangle$ then $\exists N'$ such that $M' \Downarrow \langle i, N' \rangle$ and $N \mathrel{\mathcal{R}} N'$.

Lower and convex bisimulation are as before.

The (imperative $\to$ functional) translation preserves and reflects lower and convex bisimilarity.

Properties of applicative bisimilarity

Lower applicative bisimilarity is a congruence, by Howe's method.

Convex applicative bisimilarity is a congruence in the presence of erratic nondeterminism, and [MFPS 2007] of ground amb.

But not in the presence of general amb (previous example).

In the nondeterministic setting, it is finer than may-must equivalence, e.g. Boudol-Abramsky example.

$$\begin{array}{l} \mathtt{choose}^{\perp}\ n \in \mathbb{N}.\ \langle i, \mathtt{choose}^{\perp}\ m \leqslant n.\ m \rangle \simeq_{\mathsf{may-must}} \\ \mathtt{choose}^{\perp}\ n \in \mathbb{N}.\ \langle i, \mathtt{choose}^{\perp}\ m \leqslant n.\ m \rangle\ \mathtt{or}\ \langle i, \mathtt{choose}^{\perp}\ m \in \mathbb{N}.\ m \rangle \end{array}$$

Howe's method

Howe's method, showing that applicative bisimilarity is a congruence, is elegant but mysterious.

It assumes finitary syntax, but has been adapted [CMCS 2006] for infinitary syntax.

Howe's method

Howe's method, showing that applicative bisimilarity is a congruence, is elegant but mysterious.

It assumes finitary syntax, but has been adapted [CMCS 2006] for infinitary syntax.

Can we get some understanding of this method?

Böhm trees—also represented as innocent well-bracketed strategies, in the deterministic setting—abstract away from syntactic detail.

Congruence of applicative bisimilarity says that composition of innocent well-bracketed strategies preserves applicative bisimilarity. This may (?) be easier to understand.

Moving up the type hierarchy

To model lower applicative bisimilarity we have to say what functions are definable, as we move up the type hierarchy.

This is similar to the quest for a model of sequential computation.

So far, we can characterize definable functions between strategy types: they are the exploratory functions [L & Yemane, MFPS 2009].

Cf. Kahn-Plotkin sequentiality

This may be good enough for the imperative language.

But we cannot yet characterize definability at higher-order.

Is it computable at finite types? (Cf. Loader)

Model of bisimilarity: nested simulation

A 2-nested lower simulation is a simulation contained in mutual similarity.

A 3-nested lower simulation is a simulation contained in mutual 2-nested similarity. And so through all countable ordinals.

The intersection of n-nested similarity for $n < \omega_1$ is bisimilarity.

Model of bisimilarity: suggested semantics

A nested similarity set is a set A equipped with an ω_1 sequence of preorders $\mathcal{R}_n$ where

- $\mathcal{R}_n$ is contained in the symmetrization of $\mathcal{R}_m$, for every $m < n$
- the intersection of $\mathcal{R}_n$ over all $n < \omega_1$ is the discrete relation.

$\mathcal{R}_n$ represents n-nested simulation.

Model of bisimilarity: suggested semantics

A nested similarity set is a set A equipped with an ω_1 sequence of preorders $\mathcal{R}_n$ where

- $\mathcal{R}_n$ is contained in the symmetrization of $\mathcal{R}_m$, for every $m < n$
- the intersection of $\mathcal{R}_n$ over all $n < \omega_1$ is the discrete relation.

$\mathcal{R}_n$ represents n-nested simulation.

[2010] We compute the nesting fixpoint of a monotone endofunction by

- taking the least fixpoint for $\mathcal{R}_0$, giving an equivalence class for $\mathcal{R}_1$
- take the least fixpoint for $\mathcal{R}_1$ within this class, giving an equivalence class for $\mathcal{R}_2$
- etc.

Model of bisimilarity: suggested semantics

A nested similarity set is a set A equipped with an ω_1 sequence of preorders $\mathcal{R}_n$ where

- $\mathcal{R}_n$ is contained in the symmetrization of $\mathcal{R}_m$, for every $m < n$
- the intersection of $\mathcal{R}_n$ over all $n < \omega_1$ is the discrete relation.

$\mathcal{R}_n$ represents n-nested simulation.

[2010] We compute the nesting fixpoint of a monotone endofunction by

- taking the least fixpoint for $\mathcal{R}_0$, giving an equivalence class for $\mathcal{R}_1$
- take the least fixpoint for $\mathcal{R}_1$ within this class, giving an equivalence class for $\mathcal{R}_2$
- etc.

provided these least fixpoints actually exist
and provided the intersections are nonempty.

Summary

- Infinite traces: now well understood.
- Fully abstract model of may-must testing?
- Any model of may-must testing with ground amb?
- General amb: many basic operational questions.
- After amb comes fair merge.
- Lower bisimilarity: signs of progress.
- Afterwards comes convex bisimilarity.
- Affineness
- Dataflow and call-by-need