Partially Commutative Monoids & Computational Content of Classical Logic

Hans-Dieter A. Hiep

March 21, 2018

Goals of this talk

1. Explain free partially commutative monoids
2. Show simply-typed λ-calculus and $\lambda\mathcal{C}$-calculus
3. Show classical sequent calculus with focus and polarization
4. Discuss future working areas

Interactive Theorem Proving

- Interactive theorem proving gained much attention last decades
- Lean, Coq, Isabelle, NuPRL, PVS, Mizar, HOL, AutoMath, ...
- An interactive computing system consisting of:
  - A prover (human being, helped by macros and tools)
  - A verifier (trusted kernel, sophisticated type checker)
- Allows formalization of mathematics, e.g.
  - Prime number theorem, Four colouring theorem
- Allows formalization of computer science, e.g.
  - Certified C Compiler, IsaFoR (term rewriting)
- Allows **program extraction**

Software Correctness

- Goal: software correctness of distributed systems
- Existing approaches: parametric model checking to deal with infinite systems (e.g. mCRL2, PBES)
- Our approach: interactive theorem proving
- Adventure: research proof systems that allows expression of infinite concurrent systems

- Formalizing distributed and concurrent algorithms, e.g.
  - Parallel programming semantics, Internet protocols
- Formalizing correctness properties, e.g.
  - Saftey properties, privacy properties, security properties

- What about **protocol extraction**?

Monoid overview

1. Given an algebraic definition
2. Term language by signature
3. Congruence by equational specification
4. Quotient term language by congruence
5. Elect representatives of equivalence classes
6. Find language consisting only of representatives

Partially commutative monoids

Partially commutative monoids are used to model concurrency.

Definition

An *independence relation* $I \subseteq S \times S$ is:

- an irreflexive, symmetric relation over S.

Example

Events happen at some location.
Two **independent** events happen *concurrently*.
Two **dependent** events: one *happens before* the other.

Partially commutative monoids

1. Give an algebraic definition (signature, equations)

Definition (Diekert & Métivier 1997)

A *partially commutative monoid* $\mathbb{PCM}(S, I, \cdot, 1)$ is:

monoid	$\mathbb{M}(S, \cdot, 1)$
independence relation	$I \subseteq S \times S$

such that:

partial commutation	$\forall (s,t) \in I. \quad s \cdot t = t \cdot s$

Free partially commutative monoids

Fix $I \subseteq C \times C$. PCM freely generated by a carrier set C.

Definition

Let $\mathcal{T}(C)$ be defined inductively as:

$$\mathcal{T}(C) ::= C \mid \mathcal{T}(C) \cdot \mathcal{T}(C) \mid 1$$

Definition

Let $x \cong_I y$ be the smallest congruence relation on $\mathcal{T}(C)$:

- that is associative $(x \cdot y) \cdot z \cong_I x \cdot (y \cdot z)$
- that respects identity $1 \cdot x \cong_I x \cong_I x \cdot 1$
- that commutes $x \cdot y \cong_I y \cdot x$ if $(x, y) \in I$

Free partially commutative monoids

4. Quotient term language by congruence

Definition

Let C^{I*} be the *free partially commutative monoid* that is $\mathcal{T}(C)\backslash \cong_I$.

The elements of C^{I*} are equivalence classes (called *traces*).
FPCM are also called *trace monoids*.

5. Elect representatives of equivalence classes
   We have two canonical choices:
   - Knuth's **lexicographic normal form**
   - Foata normal form

Free partially commutative monoids

Let normalization assume a total order $<$ between elements in C.
Let $\ll$ be a lexicographic order on C^* induced by $<$.

Definition

Let $\varphi : C^* \to C^{I*}$ be the canonical injection, $\varphi : x \mapsto [x]$.
We call x a linearization of $[x]$.

Definition

A lexicographic normal form $[x] \in C^{I*}$ is smallest (w.r.t. $\ll$) element $\hat{x} \in C^*$ such that $\varphi(\hat{x}) = [x]$.

Algorithm: swap adjacent elements $b \cdot a$ to $a \cdot b$ if $a < b$ and $(a, b) \in I$, until no longer possible.

Free partially commutative monoids

Example

Suppose we have the traces:

$$[b \quad a \quad d \quad c \quad b \quad a]$$

and

$$[a \quad b \quad d \quad a \quad c \quad b]$$

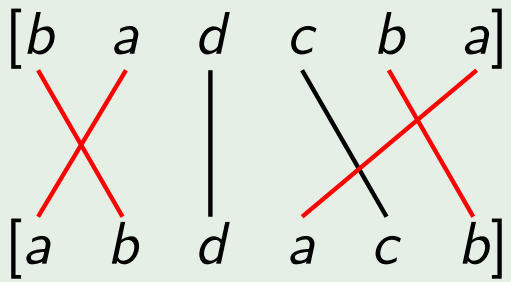
Let $(a, b) \in I$ and $(c, a) \in I$.
These are trace equivalent.
Let $a < b < c < d$.
This trace is in lexicographic normal form.

Free partially commutative monoids

Example

Suppose we have the traces:

and



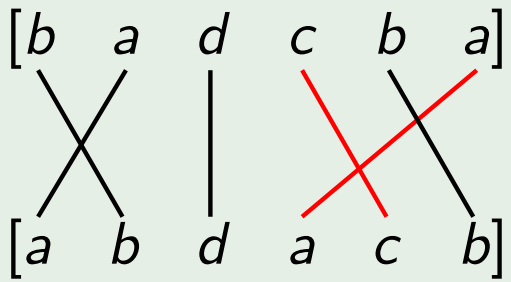
Let $(a, b) \in I$ and $(c, a) \in I$.
These are trace equivalent.
Let $a < b < c < d$.
This trace is in lexicographic normal form.

Free partially commutative monoids

Example

Suppose we have the traces:

and



Let $(a, b) \in I$ and $(c, a) \in I$.
These are trace equivalent.
Let $a < b < c < d$.
This trace is in lexicographic normal form.

Free partially commutative monoids

Example

Suppose we have the traces:

$$[b \quad a \quad d \quad c \quad b \quad a]$$

and

$$[a \quad b \quad d \quad a \quad c \quad b]$$

Let $(a, b) \in I$ and $(c, a) \in I$.
These are trace equivalent.
Let $a < b < c < d$.
This trace is in lexicographic normal form.

Free partially commutative monoids

6. Find term language consisting only of representatives

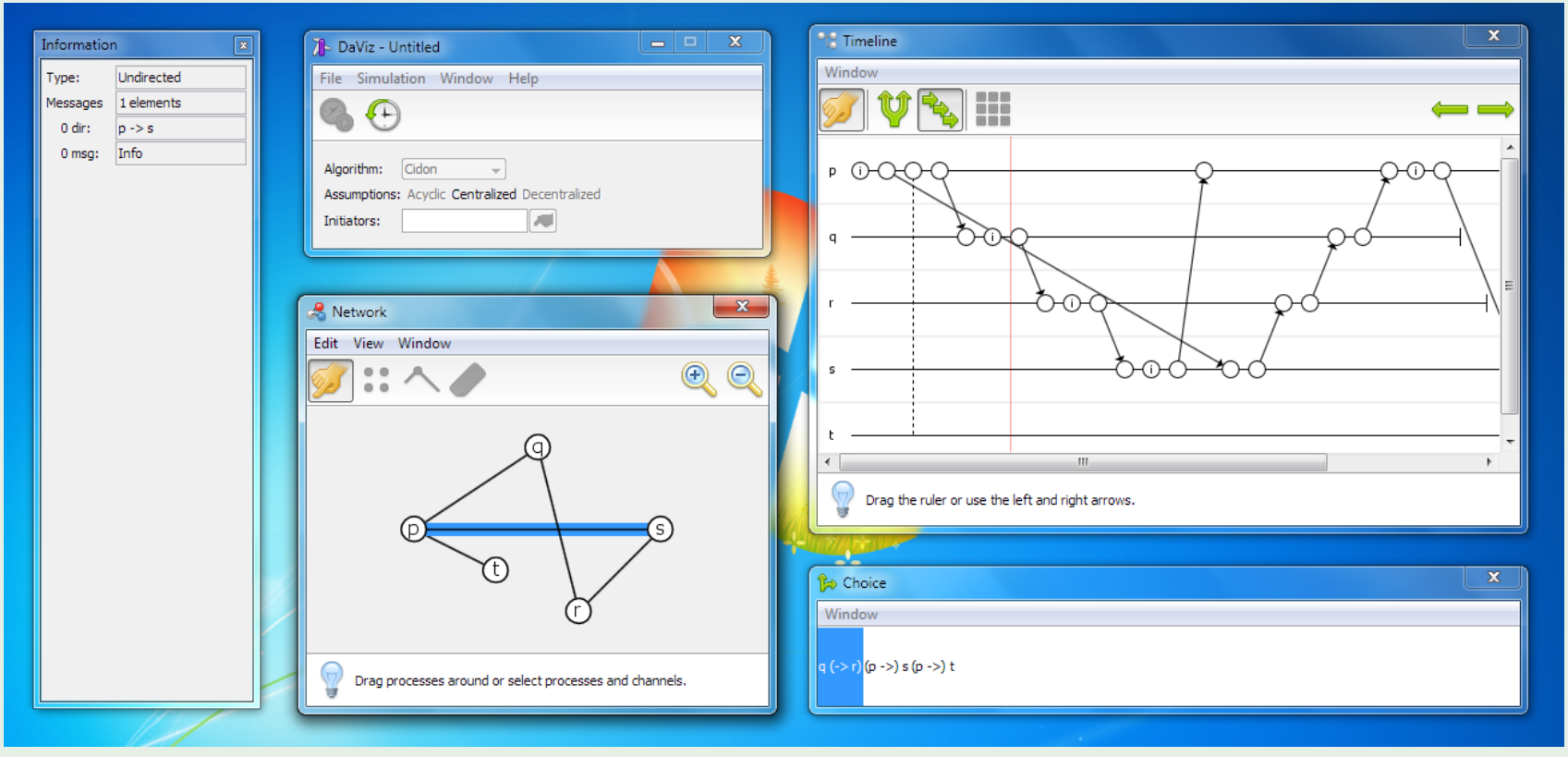
Concretely, FPCM correspond to lists that are topologically sorted.

Recap:

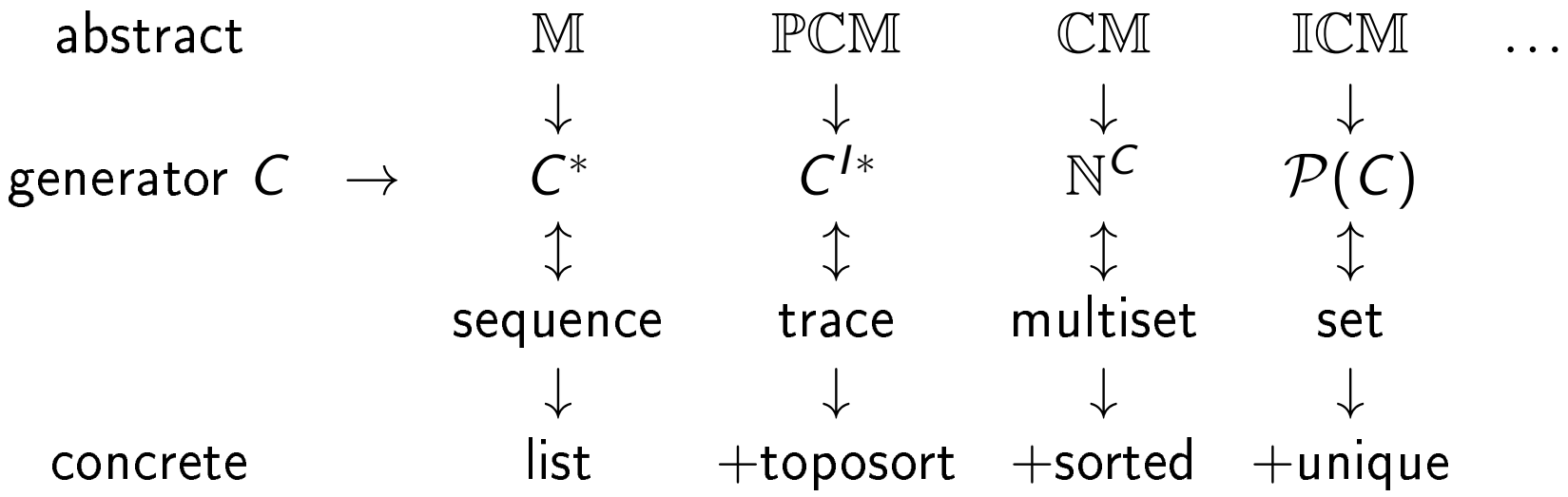
- PCM is used to model concurrency
- FPCM have unique representatives if carrier has total order

Example toy

Example



Monoid overview



Main course

1. **We consider simply-typed λ-calculus: minimal logic.**
2. **We extend λ-calculus to $\lambda\mathcal{C}$-calculus: classical logic.**
3. Sequent calculus is a meta-proof system.
4. Sequents as sets? as multisets? as PCM? as lists?
5. Focus, polarity, and generalized type connectives.
6. We see the $\mu\tilde{\mu}$-calculus: classical sequent calculus.
7. Codata is related to OOP languages.

Minimal Logic

Definition

Let $\mathcal{F}$ be the set of minimal logic formulas, defined inductively as:

$$\mathcal{F} ::= P \mid \mathcal{F} \to \mathcal{F}$$

where P is a countably infinite set of propositional variables.

$$\frac{\begin{array}{|c|}\hline \phi \\ \vdots \\ \psi \\ \hline \end{array}}{\phi \to \psi} \to_i \qquad \frac{\phi \to \psi \quad \phi}{\psi} \to_e$$

Proof system: natural deduction with two rules.

Simply-typed λ-calculus

Definition

Let M be the set of terms, defined inductively as:

$$M ::= X \mid \lambda X.M \mid MM$$

where X is a countably infinite set of variables.

$$\frac{\boxed{\begin{array}{c} x : \phi \\ \vdots \\ t : \psi \end{array}}}{\lambda x.t : \phi \to \psi}\ abs \qquad \frac{s : \phi \to \psi \quad t : \phi}{st : \psi}\ app$$

Simple type system (related to minimal logic).

Simply-typed λ-calculus

Example

Consider the derivation in minimal natural deduction

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{A \to B \quad A}{B}\,\to_e \quad B \to C}{C}\,\to_e}{A \to C}\,\to_i}{(B \to C) \to (A \to C)}\,\to_i}{(A \to B) \to (B \to C) \to (A \to C)}\,\to_i$$

that relates to the λ-term $\lambda x.\lambda y.\lambda z.y(xz)$

Classical Logic

Definition

Let $\mathcal{F}$ be the set of classical logic formulas, defined inductively as:

$$\mathcal{F} ::= P \mid \bot \mid \mathcal{F} \to \mathcal{F}$$

where P as before. We abbreviate $(\phi \to \bot)$ as $\neg\phi$.

$$\frac{\begin{array}{|c|}\hline \phi \\ \vdots \\ \psi \\ \hline \end{array}}{\phi \to \psi} \to_i \qquad \frac{\phi \to \psi \quad \phi}{\psi} \to_e \qquad \frac{\neg\neg\phi}{\phi} \neg\neg_e$$

Proof system: natural deduction with three rules.

Classical Logic

We abbreviate:

$$\begin{array}{rcl} (\bot \to \bot) & \text{as} & \top \\ (\phi \to \bot) & \text{as} & \neg\phi \\ (\phi \to \psi) \to \psi & \text{as} & \phi \vee \psi \\ \neg(\neg\phi \vee \neg\psi) & \text{as} & \phi \wedge \psi \end{array}$$

The standard rules are admissible:

$$\frac{\phi}{\phi \vee \psi}\ \vee_{i1} \qquad \frac{\psi}{\phi \vee \psi}\ \vee_{i2} \qquad \frac{\phi \vee \psi \quad \boxed{\begin{array}{c}\phi \\ \vdots \\ \chi\end{array}} \quad \boxed{\begin{array}{c}\psi \\ \vdots \\ \chi\end{array}}}{\chi}\ \vee_e$$

$$\frac{\phi \quad \psi}{\phi \wedge \psi}\ \wedge_i \qquad \frac{\phi \wedge \psi}{\phi}\ \wedge_{e1} \qquad \frac{\phi \wedge \psi}{\psi}\ \wedge_{e2} \qquad \frac{\bot}{\phi}\ \bot_e$$

λ-calculus with control

Definition ($\lambda\mathcal{C}$-calculus, Felleisen, et al. 1986)

Let M be the set of terms, defined inductively as:

$$M ::= X \mid \lambda X.M \mid MM \mid \mathcal{C}M$$

where X is a countably infinite set of variables.

$$\frac{\begin{array}{|c|}\hline x : \phi \\ \vdots \\ M : \psi \\ \hline\end{array}}{\lambda x.M : \phi \to \psi}\ abs \qquad \frac{M : \phi \to \psi \quad N : \phi}{MN : \psi}\ app \qquad \frac{M : \neg\neg\phi}{\mathcal{C}M : \phi}\ ctrl$$

Simple type system (related to classical logic).

λ-calculus with control

Example

We prove $(P \vee Q) \to (Q \vee P)$ in classical logic

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{P \vee Q}\quad \dfrac{\dfrac{\mathcal{D}}{}}{P \to Q}\to_i}{Q}\to_e \quad \dfrac{}{Q \to P}}{P}\to_e \quad \dfrac{}{\neg P}}{\bot}\to_e}{\neg\neg P}\to_i}{P}\neg\neg_e}{Q \vee P}\to_i}{(P \vee Q) \to (Q \vee P)}\to_i$$

$$\mathcal{D}: \dfrac{\dfrac{\dfrac{\neg P \quad P}{\bot}\to_e}{\neg\neg Q}\to_i}{Q}\neg\neg_e$$

that relates to the $\lambda\mathcal{C}$-term $\lambda x.\lambda y.\mathcal{C}\lambda z.z(y(x(\lambda w.\mathcal{C}\lambda v.zw)))$

Main course

1. We consider simply-typed λ-calculus: minimal logic.
2. We extend λ-calculus to $\lambda\mathcal{C}$-calculus: classical logic.
3. **Sequent calculus is a meta-proof system.**
4. **Sequents as sets? as multisets? as PCM? as lists?**
5. Focus, polarity, and generalized type connectives.
6. We see the $\mu\tilde{\mu}$-calculus: classical sequent calculus.
7. Codata is related to OOP languages.

Sequent calculus

- We have seen natural deduction proof systems.
- Drawback: implicit handling of *open* and *closed* premises.
- Corresponds to free variables in λ-terms.

Definition

Let $\Gamma, \Delta \subseteq \mathcal{F}$ be sets of formula. The *derivability relation* $\Gamma \vdash \Delta$ holds iff there exists a derivation in natural deduction with open premises in Γ and conclusion in Δ.

- Sequent calculus formalizes: $\vdash$

Sequent calculus

Remember monoids?

Definition

Let $\mathcal{S}$ be the set of sequents, defined as:

$$\mathcal{S} ::= \mathcal{F}^* \vdash \mathcal{F}^*$$

where $\mathcal{F}$ is the set of classical formula.

Structural rules:

$$\frac{\Gamma, \phi, \phi \vdash \Delta}{\Gamma, \phi \vdash \Delta} \qquad \frac{\Gamma \vdash \phi, \phi, \Delta}{\Gamma \vdash \phi, \Delta} \qquad \text{(idempotent)}$$

$$\frac{\Gamma_1, \psi, \phi, \Gamma_2 \vdash \Delta}{\Gamma_1, \phi, \psi, \Gamma_2 \vdash \Delta} \qquad \frac{\Gamma \vdash \Delta_1, \psi, \phi, \Delta_2}{\Gamma \vdash \Delta_1, \phi, \psi, \Delta_2} \qquad \text{(commutative)}$$

Sequent calculus

Logical rules:

$$\frac{}{\Gamma, A \vdash A, \Delta}\ \textit{premise} \qquad \frac{}{\Gamma, \bot \vdash \Delta}\ \textit{absurd}$$

$$\frac{\Gamma \vdash A, \Delta \quad \Gamma, B \vdash \Delta}{\Gamma, A \to B \vdash \Delta}\ \to_L \qquad \frac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A \to B, \Delta}\ \to_R$$

Sequent calculus is a meta-proof system.

- Conclusion has one formula? $\Rightarrow$ intuitionistic logic
- What if we drop idempotency? $\Rightarrow$ linear logic
- What if partially commutative? $\Rightarrow$ dependent types

(Related work: CLF, Pfenning, Watkins, et al. 2003)

Main course

1. We consider simply-typed λ-calculus: minimal logic.
2. We extend λ-calculus to $\lambda\mathcal{C}$-calculus: classical logic.
3. Sequent calculus is a meta-proof system.
4. Sequents as sets? as multisets? as PCM? as lists?
5. **Focus, polarity, and generalized type connectives.**
6. We see the $\mu\tilde{\mu}$-calculus: classical sequent calculus.
7. Codata is related to OOP languages.

Focussed sequents

We slightly change syntactical structure of sequents.

Definition (Curien & Herbelin 2000)

Let $\mathcal{S}$ be the set of sequents, defined as:

$$\begin{aligned} \mathcal{S} ::= \quad & \mathcal{F}^*;\mathcal{F} \vdash \mathcal{F}^* && \text{(term)} \\ | \quad & \mathcal{F}^* \vdash \mathcal{F};\mathcal{F}^* && \text{(context)} \\ | \quad & \mathcal{F}^* \vdash \mathcal{F}^* && \text{(command)} \end{aligned}$$

where $\mathcal{F}$ is the set of generalized formula.

Either unfocussed, or sequents have a focussed formula.

Polarities of connectives

We will use focussed sequents in two different places:

1. **As defining clauses for specifying connectives.**
2. As the objects occuring in rules of our proof and type system.

Considering generalized connectives, we distinguish two polarities:

1. **data** has clauses that define constructors
2. **class** has clauses that define observations
   (class is also called codata)

For clauses, the intuition of $\Gamma; \phi \vdash \Delta$ or $\Gamma \vdash \phi; \Delta$ is:
"assumptions act as inputs, and conclusions act as outputs."
(Downen & Ariola 2014)

Generalized type connectives

Defining clauses of data have focus on the right.
Defining clauses of class have focus on the left.

data $A \oplus B$	**data** $A \otimes B$	**data** 1	**data** 0
$A \vdash A \oplus B;$	$A, B \vdash A \otimes B;$	$\vdash 1;$	
$B \vdash A \oplus B;$			
class $A \& B$	**class** $A \parr B$	**class** $\bot$	**class** $\top$
$; A \& B \vdash A$	$; A \parr B \vdash A, B$	$; \bot \vdash$	
$; A \& B \vdash B$			

All types except for $A \parr B$ make sense intuitionistically.
"$\top$ is an abstract object with no possible observations"
$\Rightarrow$ the truth is unobservable (Benjamin)

Generalized type connectives

Functions are codata. Dual of function is data.

class $A \to B$ **where**

$A; A \to B \vdash B$

data $A - B$ **where**

$A \vdash A - B; B$

Intuition:
provide an input A and a function $A \to B$, obtain output B.

(Calling codata **class** is provocative: still many researchable questions left.)

Main course

1. We consider simply-typed λ-calculus: minimal logic.
2. We extend λ-calculus to $\lambda\mathcal{C}$-calculus: classical logic.
3. Sequent calculus is a meta-proof system.
4. Sequents as sets? as multisets? as PCM? as lists?
5. Focus, polarity, and generalized type connectives.
6. ~~We see the $\mu\tilde{\mu}$-calculus: classical sequent calculus.~~
7. **Codata is related to OOP languages.**

Object-orientation

Let's explore two equivalent formulas $A \to B$ and $(A \to \bot) \parr B$.
We will interpret them as "objects" and "processes":

class $A \to B$ **where**

$A; A \to B \vdash B$

class $A \parr B$

$; A \parr B \vdash A, B$

class $\bot$

$; \bot \vdash$

Think of "pseudo Java code":

```
interface Function { B call(A a); }
interface Splitter { A,B split(); }
interface Throwing { throw(); }
```

Object-orientation

Example

Given object that implements $A \to B$, implement $(A \to \bot) \parr B$.

- If split is called, create a channel and fork the current process.
  1. In the first process, return a function $(A \to \bot)$. When that function is called we supply the argument A to our channel, and kill our process ($\bot$).
  2. In the second process, we wait for the channel to supply A. Once we obtain A, we call the function $A \to B$, and thus obtain B. We return B.

The split method thus returns twice (one in each process). Once the first process calls $A \to \bot$, it will be killed. Only after the first process is killed will the second process 'start'.

Object-orientation

Example

Given object that implements $(A \to \bot) \parr B$, implement $A \to B$.

- If the function is called, we call split.
  1. If we obtain $(A \to \bot)$, we call it by supplying A. Our process is killed ($\bot$), so we do not need to return B.
  2. If we obtain B, we simply return it.

Hypothesis: this is a proof that the two classes are isomorphic.

Object-orientation

Example

Implement $(A \to B) \otimes (B \to A)$, axiom of linearity.

- If the function split is called, we create two channels and fork the current process.
  1. In the first process, return a function $(A \to B)$. Once that is called, we supply A to the first channel. We wait to obtain B from the second channel, returning it after it is available.
  2. Similarly, in the second process we return a function $(B \to A)$. Once that is called, we supply B to the second channel. We wait to obtain A from the first channel.

We can translate this protocol to a simulating program for single processor.

Dessert

Plan of approach:

- Write masters thesis (5 months)

Future work:

- Gather intuitive examples of codata.
- Investigate termination proofs of recursive definitions.
- Investigate productivity proofs of corecursive definitions.
- Investigate concurrent type checking.
- Investigate encoding of components (viz. Reo).
- Investigate rational/irrational components.

Dessert

There are an infinite number of computers.