From Proof Terms to Programs

An operational and quantitative study of intuitionistic Curry-Howard languages

Loïc Peyrot
18 novembre 2022
IRIF, Université Paris Cité
Functional vs imperative languages

Functional program: “what”

```ocaml
let rec factorial = function
| 1  => 1
| n  where n > 1  =>
    n * factorial (n-1)
```

Imperative program: “how”

```c
int factorial (int n) {
    int factn = 1;
    while (n >= 1) {
        factn = factn * n;
        n--;  
    }
    return factn;
}
```

Functional languages have a solid mathematical underlying theory.
At the core of functional programming are abstract models of computation. They:

- Assert fundamental properties of classes of languages.
- Influence implementations.
- Are oblivious to some implementation details.

**Our main tools**

The theory of $\lambda$-calculi and quantitative types.
What is a \( \lambda \)-calculus?

- An elementary syntax of **terms** (programs).

**Example**

In the original \( \lambda \)-calculus of Church, terms are built with three constructors:
variables \( x \), abstractions \( \lambda x.t \) and applications \( tu \).
What is a λ-calculus?

• An elementary syntax of terms (programs).

Example

In the original λ-calculus of Church, terms are built with three constructors: variables $x$, abstractions $\lambda x.t$ and applications $tu$.

• Reduction rules on terms, that represent computational progress.

Example

In Church’s λ-calculus: a unique rule $(\lambda x.t)u \rightarrow_\beta t\{x/u\}$. 
We give a meaning to programs. Two kinds of semantics are relevant for us:

**Operational semantics** is concerned with reductions on terms generated by the reduction rules.

**Denotational semantics** is concerned with general properties on terms invariant by reduction.

Different λ-calculi give rise to different semantics.
Logical systems can be seen as models of computations.

<table>
<thead>
<tr>
<th>Languages</th>
<th>Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Types</td>
<td>Propositions</td>
</tr>
<tr>
<td>Programs</td>
<td>Proofs</td>
</tr>
<tr>
<td>Evaluation</td>
<td>Cut-elimination</td>
</tr>
</tbody>
</table>

function check(n)
  
  // check if the number n is a prime
  var factor; // if the checked number is not a prime, this is its first factor
  var c;
  factor = 0;
  // try to divide the checked number by all numbers till its square root
  for (c=2; c <= Math.sqrt(n) ; c++)
  {
    if (n%c == 0) // is n divisible by c ?
      { factor = c; break }
  }
  return (factor);

  // end of check function
<table>
<thead>
<tr>
<th>Calculi</th>
<th>Intuitionistic proof systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda$-calculus</td>
<td>Natural deduction (ND)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Atomic $\lambda$-calculus</td>
<td>Open deduction</td>
</tr>
<tr>
<td>(Node replication)</td>
<td>Guglielmi, Gundersen, Parigot (2010)</td>
</tr>
<tr>
<td>Gundersen, Heijltjes &amp; Parigot (2012)</td>
<td></td>
</tr>
<tr>
<td>$\lambda$-calculus with gen. applications</td>
<td>ND with gen. elimination</td>
</tr>
</tbody>
</table>
We look into semantical properties of reduction with node replication or generalized applications, both:

<table>
<thead>
<tr>
<th>Qualitative</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a) Does a given term normalize?</td>
<td></td>
</tr>
<tr>
<td>b) Given two evaluation strategies, do they both normalize or diverge for a same term?</td>
<td></td>
</tr>
</tbody>
</table>
This work

We look into semantical properties of reduction with node replication or generalized applications, both:

**Qualitative**

a) Does a given term normalize?
b) Given two evaluation strategies, do they both normalize or diverge for a same term?

**Quantitative**

c) What is the reduction length of a given term to normal form?
d) Does an evaluation strategy normalize in less steps than another?
Intersection types capture normalization

\[ \lambda x.xxx \] is typable with intersection types.

\[ \tau \rightarrow \sigma \land \tau \rightarrow \sigma \rightarrow \tau \]

\[ \lambda x.xxx \]
## Idempotent and non-idempotent intersection types

<table>
<thead>
<tr>
<th>Idempotent</th>
<th>Non-idempotent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coppo &amp; Dezani (80’s)</td>
<td>Gardner, Kfoury (90’s), de Carvalho (2007)</td>
</tr>
<tr>
<td>(\tau \land \tau = \tau)</td>
<td>(\tau \land \tau \neq \tau)</td>
</tr>
<tr>
<td>Qualitative analysis</td>
<td>Quantitative analysis</td>
</tr>
</tbody>
</table>

\[\lambda f. \lambda x. fxx\]

![Idempotent Example](image)

![Non-idempotent Example](image)
Node Replication
Different Curry-Howard notions of substitution

<table>
<thead>
<tr>
<th>Substitution kind</th>
<th>Logical framework</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Substitution</td>
<td>Natural Deduction</td>
</tr>
<tr>
<td>Linear Substitution</td>
<td>Linear Logic</td>
</tr>
<tr>
<td>Node Replication</td>
<td>Open Deduction</td>
</tr>
</tbody>
</table>

How is normalization affected by node replication (qualitatively and quantitatively)?
Node replication

Duplication of terms constructor by constructor. Enables optimizations by keeping more subterms shared.

(Node replication)

(Full substitution)
1. Define a simple calculus with node replication (called $\lambda R$).
2. Define different evaluation strategies in the calculus.
3. Give a quantitative model for these strategies.
4. Prove observational equivalence between these strategies.
Firing substitution in the $\lambda R$-calculus

(Terms)  \( t, u ::= x \mid \lambda x.t \mid tu \mid t[x/u] \mid t[x//\lambda y.u] \)

Definition (B-rule)

\[(\lambda x.t)u \rightarrow_B t[x/u] \]

Some reductions are blocked by ES:

\[(\lambda x.t)[y/v]u \not\rightarrow_B \]
Reduction at a distance

Reduction can be recovered by adding structural permutations.

\[(\lambda x.t)[y/v]u \rightarrow_{\rho} ((\lambda x.t)u)[y/v] \rightarrow_{B} t[x/u][y/v]\]

Our approach: distance

\[B + \text{needed permutations} = dB:\]

\[L\langle\lambda x.t\rangle u \rightarrow_{dB} L\langle t[x/u]\rangle, \text{where}\ L = \diamond [x_1/u_1] \ldots [x_n/u_n].\]
The $\lambda_R$-calculus

(Terms) $t, u ::= x \mid \lambda x.t \mid tu \mid t[x/u] \mid t[x//\lambda y.u]$
The $\lambda R$-calculus

(Terms) $t, u ::= x | \lambda x.t | tu | t[x/u] | t[x//\lambda y.u]$

$L\langle \lambda x.t \rangle u \rightarrow_{dB} L\langle t[x/u] \rangle$

} Firing substitution

} Substitution
The $\lambda R$-calculus

(Terms) $t, u ::= x \mid \lambda x.t \mid tu \mid t[x/u] \mid t[x//\lambda y.u]$

\[
\begin{align*}
L\langle \lambda x.t \rangle u & \mapsto dB & L\langle t[x/u] \rangle \\
t[x/L\langle y \rangle] & \mapsto \text{var} & L\langle t\{x/y\} \rangle \\
t[x/L\langle uv \rangle] & \mapsto \text{app} & L\langle t\{x/yz\}[y/u][z/v] \rangle \\
t[x/L\langle \lambda y.u \rangle] & \mapsto \text{dist} & L\langle t[x//\lambda y.z[z/u]] \rangle \\
t[x//\lambda y.u] & \mapsto \text{abs} & L\langle t\{x/\lambda y.p\} \rangle \\
\end{align*}
\]

where $u \rightarrow^* L\langle p \rangle$ and $y \notin \text{fv}(L)$. 

\} Firing substitution
\} Substitution
• An optimization of call-by-need (CbNeed).
• Can be implemented by node replication.
• Only duplicates the skeleton of abstractions.
• The skeleton is the path from the topmost abstraction $\lambda y$ to the occurrences of $y$.
• The complement of the skeleton stays shared.
• This avoids some duplication of computations.
Example of graphical fully lazy duplication

\[ \lambda y. y(II) = \lambda y. yz + [z/II] \]

\(\text{skeleton}\)

\(\text{sharing}\)
Example of fully lazy duplication in $\lambda R$

Full laziness can be implemented in $\lambda R$.

$$(\lambda x.xx)(\lambda y.y(II)) \rightarrow_{dB} (xx)[x/\lambda y.y(II)]$$

$$\rightarrow_{dist} (xx)[x//\lambda y.z[z/y(II)]]$$

$$\rightarrow_{app} (xx)[x//\lambda y.(z_1z_2)[z_1/y][z_2/II]]$$

$$\rightarrow_{var} (xx)[x//\lambda y.(yz_2)[z_2/II]]$$

$$\rightarrow_{abs} ((\lambda y.yz_2)(\lambda y.yz_2))[z_2/II]$$
Comparison with the atomic $\lambda$-calculus

<table>
<thead>
<tr>
<th>$\lambda R$-calculus</th>
<th>Atomic $\lambda$-calculus</th>
</tr>
</thead>
<tbody>
<tr>
<td>with Delia Kesner, Daniel Ventura (FoSSaCS 2021)</td>
<td>Gundersen, Heijltjes &amp; Parigot (2012)</td>
</tr>
<tr>
<td>Non-linear variables</td>
<td>Linear variables</td>
</tr>
<tr>
<td>Distance</td>
<td>Independent permutations</td>
</tr>
<tr>
<td>Focuses on programming languages</td>
<td>Focuses on logical systems</td>
</tr>
</tbody>
</table>
We define (weak-head) call-by-name (CbN) and CBNeed strategies of $\lambda R$.

- Our CbN simulates full substitution in the $\lambda$-calculus.
- Our CbNeed is fully lazy:
  - memoization,
  - need contexts, and
  - skeleton extraction.
Two different semantic for splitting

**Big-step (one of the rules)**

\[

t \downarrow^{\theta \cup \{x\}} L \langle s \rangle \\
\lambda x. t \downarrow^{\theta} L \langle \lambda x. s \rangle
\]

**Small-step (one of the rules)**

\[
t[x/\lambda z.u] \mapsto^{y}_{\text{dist}} t[x/\lambda z.x'[x'/u]] \text{ where } y \in \text{fv}(u)
\]

**Theorem**

The two semantics are equivalent, and give the correct splitting.
Small-steps skeleton extraction is more flexible

When considering skeletons of terms with ES, the big-steps semantics may cause inefficency.

Example

Let $\lambda x. t = \lambda x. (\lambda y. y[x'/x])z$.

- $t \Downarrow \{x\} ((\lambda y. y)z')[z'/z]$, but:
- $w[w/\lambda x. t] \rightarrow^* (\lambda x. w)[w/(\lambda y. y)z]$ (in two steps)
The quantitative type system $\cap R$

Some of the typing rules:

(AX) $x : [\sigma] \vdash x : \sigma$

(ANS) $\emptyset \vdash \lambda x . t : \alpha$

(ES) $\Gamma; x : [\tau_i]_{0 \leq i \leq n} \vdash t : \sigma \quad \Delta_1 \vdash u : \tau_1 \quad \cdots \quad \Delta_n \vdash u : \tau_n$

$\Gamma \uplus \Delta_1 \uplus \cdots \uplus \Delta_n \vdash t[x/u] : \sigma$
Full laziness is an operational feature, not a semantical one.

NR = Node replication  FS = Full substitution
Characterization of normalization by typability

Contribution

quantitative type system $\cap R$

call-by-name (NR)  
fully lazy call-by-need (NR)

call-by-name (FS)  
call-by-need (FS)

NR = Node replication   FS = Full substitution
• Usually, in intuitionistic calculi, the size of the non-idempotent type derivation decreases at each step.

• In $\lambda R$, rules $\text{app}$ and $\text{dist}$ adds fresh variables that makes the size of the derivation grow.

\[ t[x/u_1u_2] \rightarrow_{\text{app}} t[x/x_1x_2][x_1/u_1][x_2/u_2] \]

• We define a decreasing measure on type derivations, enabling a combinatorial proof of normalization.
At every step of reduction, the measure on type derivations decreases.

With permutations: not every step consumes resources.

With distance: every step consumes resources.
Qualitative questions

a) Does a given term normalize?
b) Given two evaluation strategies, do they both normalize or diverge for a same term?

Answers:
a) If and only if it is typable in system $\cap R$.
b) CbN and CbNeed, with full substitution or node replication all normalize on the same terms.
### Quantitative questions

**c)** What is the reduction length of a given term to normal form?

**d)** Does an evaluation strategy normalize in less steps than another?

**Answers:**

**c)** The measure gives an upper bound on the number of reduction steps.

**d)** Full laziness reduces the length of reduction w.r.t. full substitution.
Generalized Applications
Generalized applications (GA) are a Curry-Howard interpretation of natural deduction with generalized elimination rules.

<table>
<thead>
<tr>
<th>Original calculi</th>
<th>Distant variants</th>
</tr>
</thead>
<tbody>
<tr>
<td>CbN: $\Lambda J$ (Joachimski &amp; Matthes, 2000)</td>
<td>$\lambda J_n$ (new)</td>
</tr>
<tr>
<td>CbV: $\Lambda J_v$ (Espírito Santo, 2020)</td>
<td>$\lambda J_v$ (new)</td>
</tr>
</tbody>
</table>

CbN: call-by-name
CbV: call-by-value
The syntax of terms for generalized applications

(Values) \( v ::= x | \lambda x. t \)

(Terms) \( t, u, r ::= v | t(u, x.r) \)

\[
\frac{\Gamma \vdash x : A \quad \text{AX}}{\Gamma, x : A \vdash x : A}
\]

\[
\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x. t : A \rightarrow B \quad \rightarrow_i}
\]

\[
\frac{\Gamma \vdash t : A \rightarrow B \quad \Gamma \vdash u : A \quad \Gamma, x : B \vdash r : C}{\Gamma, x : B \vdash r : C \quad \rightarrow_e}
\]
All and only applications are shared

<table>
<thead>
<tr>
<th>Shared?</th>
<th>Variables</th>
<th>Abstractions</th>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>ES</td>
<td>$r[x/y]$</td>
<td>$r[x/\lambda y.t]$</td>
<td>$r[x/tu]$</td>
</tr>
<tr>
<td>GA</td>
<td>no</td>
<td>no</td>
<td>$t(u, x.r)$</td>
</tr>
</tbody>
</table>

First intuition (not completely right)

$t(u, x.r) \approx \text{let } x = tu \text{ in } r \approx r[x/tu]$
Rewriting in the original CbN calculus with GA

A $\beta$-rule with meta-level substitutions

$$(\lambda x.t)(u, y.r) \rightarrow_\beta r\{y/t\{x/u}\}$$

- Generalizes $\beta$-reduction in the $\lambda$-calculus.

A commutative conversion $\pi$

$$t(u, x.r)(u', y.r') \rightarrow_\pi t(u, x.r(u', y.r'))$$

- Moves the leftmost redex on top of the term.
Failure of CbN subject reduction for $\pi$

We give a quantitative type system $\cap J$ for CbN reduction of GA.

### Subject reduction/expansion in a quantitative type system

- **Weighted subject reduction:** $t_1 \rightarrow t_2 \rightarrow \ldots \rightarrow t_n$ \[\Gamma \vdash t : \tau\]
- **Subject expansion:** $t_1 \leftarrow \ldots \leftarrow t_n$ \[\Gamma \vdash t : \tau\]

But with $\pi$: subject reduction in the quantitative system fails.

**Question**

e) Can we define a CbN calculus with generalized applications compatible with a quantitative model?

Joint work with Delia Kesner and José Espírito Santo, FoSSaCS 2022.
Permutations are necessary

We cannot remove $\pi$-permutations without changing normalization, because $\pi$-permutations are useful to unblock beta-reduction.

$$z(u_1, y_1. x . x)(u_2, y_2. y_2) \rightarrow_\beta$$
Permutations are necessary

We cannot remove $\pi$-permutations without changing normalization, because $\pi$-permutations are useful to unblock beta-reduction.

$$z(u_1, y_1 \cdot \lambda x. x)(u_2, y_2 \cdot y_2) \rightarrow_\pi z(u_1, y_1 \cdot (\lambda x. x)(u_2, y_2 \cdot y_2))$$

$$\rightarrow_\beta z(u_1, y_1 \cdot u_2)$$
A new CbN calculus

- We consider instead the permutation rule $p_2$:

$$t(u, y.\lambda x.r) \rightarrow_{p_2} \lambda x.t(u, y.r)$$

- We define a distant calculus $\lambda J_n$ based on $p_2$ and using a single distant rule $d_\beta$.

- Unlike $\Lambda J$, this calculus is compatible with the quantitative type system:
  - a) Typability characterizes strong normalization.
  - c) The size of type derivations gives an upper bound on the length of reduction and size of normal forms.
Comparison of the semantics of the CbN calculi

b) Given two evaluation strategies, do they both normalize or diverge for a same term?

d) Does an evaluation strategy normalize in less steps than another?

Answers:

• b) Strong normalization of $\lambda J_n$ and $\Lambda J$ correspond.
• d) The quantitivities are incomparable.
Definition (Distant contexts)

\[ D ::= \diamondsuit \mid t_1(t_2, x.D) \]

What makes \( \lambda J_n \) and \( \Lambda J \) different? Compare:

(\( \lambda J_n \)) \quad D(\lambda x.t)(u, y.r) \rightarrow_{d\beta} r\{y/D(t\{x/u}\})

Duplication or erasure of \( D \)

(\( \Lambda J \)) \quad D(\lambda x.t)(u, y.r) \rightarrow^{*}_{\pi} \beta D(r\{y/t\{x/u}\})

Sharing of \( D \)
We want to relate strong normalization in GA and the λ-calculus (with explicit substitutions).

**Reminder: initial (wrong) intuition**

\[ t(u, x.r) \approx r[x/tu] \]

But the semantics **differs**.

**Example**

Let \( \delta = \lambda x.xx \) and \( \delta_j = \lambda x.x(x, z.z) \).

The terms \( \delta_j(\delta_j, x.\lambda y.y) \) and \( (\lambda y.y)[x/\delta\delta] \) seem to correspond. But in CbN, only the first one is strongly normalizing.
Theorem

Translations preserve strong normalization both way.
Does the operational semantics of generalized applications enable to capture semantical properties?

We look at:

- A perpetual strategy.
- A normalizing strategy.
- Solvability (FSCD 2022).
The Cbn and CbV original calculi

Call-by-name (Joachimski & Matthes):

\[(\lambda x.t)(u, y.r) \rightarrow_\beta r\{y/t\{x/u}\}\]  \(\quad\) \(\Lambda J\)
\[t(u, x.r)(u', y.r') \rightarrow_\pi t(u, x.r(u', y.r'))\]

Call-by-value (Espírito Santo):

\[(\lambda x.t)(u, y.r) \rightarrow_\beta v r\{y\|t\{x\|u\}\}\]  \(\quad\) \(\Lambda J_v\)
\[t(u, x.r)(u', y.r') \rightarrow_\pi t(u, x.r(u', y.r'))\]

Definition (CbV substitution)

\[t\{x\|D\langle v\rangle\} = D\langle t\{x/v\}\rangle\]
The new CbN and CbV distant calculi

\[ \text{CbN, based on p2} \]

\[ D\langle \lambda x.t \rangle (u, y.r) \rightarrow_{d\beta} \ r\{y/D\langle t\{x/u}\rangle\} \]

\[ \lambda J_n \]

\[ \text{CbV, based on } \pi \]

\[ D\langle \lambda x.t \rangle (u, y.r) \rightarrow_{d\beta_v} \ D\langle r\{y\parallel t\{x\parallel u\}\} \rangle \]

\[ \lambda J_v \]
A benefit of using generalized applications

- In the CbN $\lambda$-calculus, the leftmost-outermost reduction is normalizing.
- Giving a normalizing strategy for a CbV calculus is in general much more difficult (Leberle, 2021).

In the framework of generalized applications:

- We give a simple normalizing strategy for CbV.
- This strategy reduces redexes in the leftmost-outermost order.
A normalizing leftmost-outermost CbV strategy (for $\Lambda J_v$)

Normal forms

$$NF ::= x \mid \lambda x. NF \mid x(NF, y. NF)$$

Base rules

$$\beta_v + \pi$$

Contextual rules

<table>
<thead>
<tr>
<th>$t \rightarrow t'$</th>
<th>$u \rightarrow u'$</th>
<th>$r \rightarrow r'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda x.t \rightarrow \lambda x.t'$</td>
<td>$x(u, y.r) \rightarrow x(u', y.r)$</td>
<td>$x(u, y.r) \rightarrow x(u, y.r')$</td>
</tr>
</tbody>
</table>

Taking the same normal forms and inductive rules, we can obtain a CbN normalizing strategy.
Call-by-name solvability for generalized applications

**Contribution**
Characterizations of CbN solvability in $\lambda J_n$ and $\Lambda J$.

**Definition**
$t$ is CbN solvable: $\exists H, D$ such that
$H \langle t \rangle \rightarrow_{\lambda J_n} D \langle \lambda x.x \rangle$.

**Theorem**
Translations to and from the $\lambda$-calculus preserve solvability.
What about CbV solvability?

- Normalizable terms should should all be meaningful.
- But Plotkin’s CbV calculus is defective:
- The term \((\lambda x.\delta)(yy)\delta\) (for instance) has no denotation but is in normal form.
CbV solvability for generalized applications

Contribution
Characterizations of CbV solvability in $\lambda J_v$ and $\Lambda J_v$.

Definition
$t$ is CbV solvable: $\exists H$ such that $H\langle t \rangle \rightarrow_{\lambda J_v} \lambda x . x$. 
Alternative CbV operational characterizations

Operational characterizations of CbV solvability were already given for two other calculi.

\[ \lambda_{\text{vsub}} \text{ (Accattoli & Paolini, 2012)} \]

Uses explicit substitutions and distance:

\[
(\lambda x.\delta)(yy)\delta \rightarrow_{d\beta_v} (\delta\delta)[x/yy] \odot^2_{d\beta_v}
\]

\[ \lambda_{\text{v}}^\sigma \text{ (Carraro & Guerrieri, 2014)} \]

Adds permutations to Plotkin’s calculus:

\[
(\lambda x.\delta)(yy)\delta \rightarrow_{\sigma_1} (\lambda x.\delta\delta)(yy) \odot_{\beta_v}
\]
Comparing CbV solvability in different frameworks

Theorem

Translations between GA and the λ-calculus (with explicit substitutions) preserve CbV solvability both ways.
Comparing CbV solvability in different frameworks

**Theorem**

Translations between GA and the \( \lambda \)-calculus (with explicit substitutions) preserve CbV solvability both ways.

We can compare the solving evaluation strategies:

<table>
<thead>
<tr>
<th></th>
<th>GA</th>
<th>( \lambda_v^\sigma )</th>
<th>( \lambda_v^{\text{sub}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple normal forms</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Meta-level substitutions</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Stuck reductions</td>
<td>No</td>
<td>( (\lambda x.x)(yy) \not\rightarrow )</td>
<td>( x[x/yy] \not\rightarrow )</td>
</tr>
<tr>
<td>Moggi’s identity</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

**Definition (Moggi’s identity)**

For any term \( u \), \( (\lambda x.x)u \not\rightarrow u \).
Conclusion
Contributions

• A new calculus for node replication.
• CbN and fully lazy CbNeed strategies based on node replication.
• Quantitative models for these strategies.
• CbN and CbV distant calculi with GA.
• Operational characterizations of solvability and weak normalization in GA.
• Quantitative models for CbN and CbV generalized applications.
Further works

Short term:

• Exact bounds with tight type systems.
• Abstract machines for full laziness.

Long term:

• Understand the correct notion of meaningless term in CbV equipped with a genericity lemma.
• Classical calculi to capture control operators in generalized applications and node replication.
• Fully abstract CbV models.
Thank you for your attention!