From Implicit Complexity to Quantitative Resource Analysis

**Bounded Linear Logic and Linear Dependency**

*Ugo Dal Lago*

(Joint work with *Martin Hoffman* and *Marco Gaboardi*)

PhD Open, University of Warsaw, June 25-27, 2015
Part I

Bounded Linear Logic
Comparing ICC Systems.

- “Traditional” definitions of equivalence between programming languages do not make much sense.
- The sets of representable first-order functions are almost the same.
  - The systems under consideration are anyway sound and complete characterization of polynomial time computable functions.
    - But we could have higher-order functions or not.
    - Some systems have iteration, others have recursion.
    - ...

- How can we express that one system $S$ is intensionally more powerful than another one, $\mathcal{P}$?
- One possibility is the one advocated by Felleisen [Felleisen95]:
  - $S$ is (strictly) more expressive than $\mathcal{P}$ iff $S$ is a syntactical extension of $\mathcal{P}$ and the constructs characterizing $S$ are not derivable in $\mathcal{P}$.
  - Too fine-grained for our purposes.
Comparing ICC Systems.

- “Traditional” definitions of equivalence between programming languages do not make much sense.
- The sets of representable first-order functions are almost the same.
  - The systems under consideration are anyway sound and complete characterization of polynomial time computable functions.
  - But we could have higher-order functions or not.
  - Some systems have iteration, others have recursion.
  - ...
- How can we express that one system $S$ is intensionally more powerful than another one, $P$?
- One possibility is the one advocated by Felleisen [Felleisen95]:
  - $S$ is (strictly) more expressive than $P$ iff $S$ is a syntactical extension of $P$ and the constructs characterizing $S$ are not derivable in $P$.
  - Too fine-grained for our purposes.
Comparing ICC Systems.

- “Traditional” definitions of equivalence between programming languages do not make much sense.
- The sets of representable first-order functions are almost the same.
  - The systems under consideration are anyway sound and complete characterization of polynomial time computable functions.
  - But we could have higher-order functions or not.
  - Some systems have iteration, others have recursion.
  - ...

How can we express that one system $S$ is intensionally more powerful than another one, $P$?

- One possibility is the one advocated by Felleisen [Felleisen95]:
  - $S$ is (strictly) more expressive than $P$ iff $S$ is a syntactical extension of $P$ and the constructs characterizing $S$ are not derivable in $P$.
  - Too fine-grained for our purposes.
Comparing ICC Systems.

- “Traditional” definitions of equivalence between programming languages do not make much sense.
- The sets of representable first-order functions are almost the same.
  - The systems under consideration are anyway sound and complete characterization of polynomial time computable functions.
  - But we could have higher-order functions or not.
  - Some systems have iteration, others have recursion.
  - ...

- How can we express that one system $S$ is intensionally more powerful than another one, $P$?
- One possibility is the one advocated by Felleisen [Felleisen95]:
  - $S$ is (strictly) more expressive than $P$ iff $S$ is a syntactical extension of $P$ and the constructs characterizing $S$ are not derivable in $P$.
  - Too fine-grained for our purposes.
Comparing ICC Systems.

- “Traditional” definitions of **equivalence** between programming languages do not make much sense.
- The sets of representable first-order **functions** are almost the same.
  - The systems under consideration are anyway sound and complete characterization of polynomial time computable functions.
  - But we could have higher-order functions or not.
  - Some systems have iteration, others have recursion.
  - ...

- How can we express that one system $S$ is **intensionally** more powerful than another one, $\mathcal{P}$?
- One possibility is the one advocated by Felleisen [Felleisen95]:
  - $S$ is (strictly) more expressive than $\mathcal{P}$ iff $S$ is a syntactical extension of $\mathcal{P}$ and the constructs characterizing $S$ are not **derivable** in $\mathcal{P}$.
  - Too fine-grained for our purposes.
Compositional Embeddings.

- We here adopt another criterion, which many others have previously considered.
- $\mathcal{S}$ is at least as expressive as $\mathcal{P}$ iff there exists a compositional embedding of $\mathcal{P}$ into $\mathcal{S}$.
- What is a compositional embedding? Well... it depends.
  1. On the syntax of $\mathcal{S}$ and $\mathcal{P}$: the embedding should act homeomorphically on $\mathcal{P}$’s syntax.
  2. On the (operational?) semantics of $\mathcal{S}$ and $\mathcal{P}$: the embedding should map any $\mathcal{P}$ program to an equivalent one.
- Usually, the languages of $\mathcal{S}$ and $\mathcal{P}$ are defined inductively.
  - The meaning of 1. is reasonably clear.
- What about the semantics?
  - We prove 2. by studying denotational semantics.
The Current Situation.

- Let’s consider some of the known characterizations of polynomial time functions: LFPL \[Hofmann99\], BC \[BellantoniCook92\], and LLL \[Girard97\].

- It seems that LFPL and BC are not intensionally comparable:
  - On one side a system of non-size-increasing functions. On the other side a system which is complete for polytime functions.
  - But no formal result.

- LLL vs. BC: the same.
  - Murawski and Ong showed that there cannot be any embedding (satisfying certain properties) of BC into LAL.
  - On the other hand, LAL is a system for higher-order functions, while BC is a function algebra...

- What about BLL?
  - One of the very first characterizations of the polytime functions \[GirardScedrovScott92\].
  - Hasn’t received too much attention since then.
Let’s consider some of the known characterizations of polynomial time functions: LFPL \cite{Hofmann99}, BC \cite{BellantoniCook92}, and LLL \cite{Girard97}.

It seems that LFPL and BC are not intensionally comparable:

- On one side a system of non-size-increasing functions. On the other side a system which is complete for polytime functions.
- But no formal result.

LLL vs. BC: the same.

- Murawski and Ong showed that there cannot be any embedding (satisfying certain properties) of BC into LAL.
- On the other hand, LAL is a system for higher-order functions, while BC is a function algebra...

What about BLL?

- One of the very first characterizations of the polytime functions \cite{GirardScedrovScott92}.
- Hasn’t received too much attention since then.
The Current Situation.

- Let’s consider some of the known characterizations of polynomial time functions: LFPL [Hofmann99], BC [BellantoniCook92], and LLL [Girard97].

- It seems that LFPL and BC are not intensionally comparable:
  - On one side a system of non-size-increasing functions. On the other side a system which is complete for polytime functions.
  - But no formal result.

- LLL vs. BC: the same.
  - Murawski and Ong showed that there cannot be any embedding (satisfying certain properties) of BC into LAL.
  - On the other hand, LAL is a system for higher-order functions, while BC is a function algebra...

- What about BLL?
  - One of the very first characterizations of the polytime functions [GirardScedrovScott92].
  - Hasn’t received too much attention since then.
Our Results.

- A natural generalization of BLL, called QBAL.
- A soundness theorem for QBAL with respect to polynomial time.
- A compositional embedding of Hofmann’s LFPL into QBAL.
- A compositional embedding of Leivant’s RRW into QBAL.
  - BC itself can be easily embedded into RRW.
Our Results.

- A natural generalization of BLL, called QBAL.
- A soundness theorem for QBAL with respect to polynomial time.
- A compositional embedding of Hofmann’s LFPL into QBAL.
- A compositional embedding of Leivant’s RRW into QBAL.
  - BC itself can be easily embedded into RRW.
Bounded Linear Logic.

- It is a refinement on (intuitionistic) linear logic:

\[
\begin{align*}
!_n A & \to 1 \\
!_{1+n} A & \to A \\
!_{n+m} A & \to !_n A \otimes !_m A \\
!_{nm} A & \to !_n !_m A
\end{align*}
\]

where \( n \) and \( m \) are natural numbers.

- More generally, \( n \) ad \( m \) could be polynomials (on possibly many variables) and not just natural numbers.

- Moreover, \( ! \) can act as a binder for resource variables: \( !_x p A \). As an example, the following is an axiom:

\[
!_{x<p+q} A \to !_x p A \otimes !_y q A[x \leftarrow y + p]
\]

- Intuitively:

\[
!_{x<p} A \sim A[x \leftarrow 0] \otimes A[x \leftarrow 1] \otimes \ldots \otimes A[x \leftarrow p - 1].
\]
Bounded Linear Logic.

- It is a refinement on (intuitionistic) linear logic:

\[
\begin{align*}
!_n A & \to 1 \\
!_{1+n} A & \to A \\
!_{n+m} A & \to !_n A \otimes !_m A \\
!_{nm} A & \to !_n !_m A
\end{align*}
\]

where \( n \) and \( m \) are natural numbers.

- More generally, \( n \) ad \( m \) could be polynomials (on possibly many variables) and not just natural numbers.

- Moreover, \( ! \) can act as a binder for resource variables: \( !_{x<p} A \). As an example, the following is an axiom:

\[
!_{x<p+q} A \to !_x A \otimes !_y A[x \leftarrow y + p]
\]

- Intuitively:

\[
!_{x<p} A \sim A[x \leftarrow 0] \otimes A[x \leftarrow 1] \otimes \ldots \otimes A[x \leftarrow p - 1].
\]
Bounded Linear Logic.

- It is a refinement on (intuitionistic) linear logic:

\[
\begin{align*}
!_n A & \to 1 \\
!_{1+n} A & \to A \\
!_{n+m} A & \to !_n A \otimes !_m A \\
!_{nm} A & \to !_n !_m A
\end{align*}
\]

where \( n \) and \( m \) are natural numbers.

- More generally, \( n \) ad \( m \) could be polynomials (on possibly many variables) and not just natural numbers.

- Moreover, \( ! \) can act as a binder for resource variables: \( !_{x<p} A \). As an example, the following is an axiom:

\[
!_{x<p+q} A \to !_x A \otimes !_{y<q} A[x \leftarrow y + p]
\]

- Intuitively:

\[
!_{x<p} A \sim A[x \leftarrow 0] \otimes A[x \leftarrow 1] \otimes \ldots \otimes A[x \leftarrow p-1].
\]
Quantified Bounded Affine Logic.

- Usual, second order quantification is available in QBAL.
  - It was available in BLL, too.
- There is another form of quantification in QBAL: universal and existential quantification on resource variables:
  \[ \exists (\overline{x}) : \mathcal{C}.A \]
  \[ \forall (\overline{x}) : \mathcal{C}.A \]

  where \( \mathcal{C} \) is a set of constraints.

- Example:
  \[ \exists (x, y) : \{x \leq z^2, y \leq x\}.!_{xy}z.A \]

  Notice that the constraints in \( \{x \leq z^2, y \leq x\} \) enforce a polynomial upper bound on both \( x \) and \( y \): \( x \leq z^2 \) and \( y \leq x \leq z^2 \).
  - Not by coincidence!

- Sequents have the form \( \Gamma \vdash \mathcal{C}.A \).
- Rules for first-order quantifier are standard.
- The rules coming from BLL leaves \( \mathcal{C} \) unchanged.
Rules: Some Examples

- Axiom:
  \[
  \frac{A \sqsubseteq C \; B}{A \vdash_C B} \quad A
  \]

- Linear arrow:
  \[
  \frac{\Gamma \vdash_C A \quad \Delta, B \vdash C}{\Gamma, \Delta, A \rightarrow B \vdash C} \quad L\rightarrow \\
  \frac{\Gamma, A \vdash_C B}{\Gamma \vdash_C A \rightarrow B} \quad R\rightarrow
  \]

- Promotion:
  \[
  \frac{A_1, \ldots, A_n \vdash_C B \quad \mathcal{D}, x < p \models C \quad x \notin \text{FV}(\mathcal{D}) \quad p \sqsubseteq \mathcal{D} q_i}{!x < q_1 A_1, \ldots, !x < q_n A_n \vdash_{\mathcal{D}} !x < p B} \quad P!
  \]

- Existential quantification:
  \[
  \frac{\Gamma \vdash_C A\{\bar{p}/x\} \quad C \models \mathcal{D}\{\bar{p}/x\}}{\Gamma \vdash_C \exists x : \mathcal{D}.A} \quad R\exists x
  \]
  \[
  \frac{\Gamma, A \vdash_C \cup_{\mathcal{D}} C \quad \bar{x} \notin \text{FV}(\Gamma) \cup \text{FV}(C) \cup \text{FV}(\mathcal{C})}{\Gamma, \exists \bar{x} : \mathcal{D} : A \vdash_C C} \quad L\exists x
  \]
Rules: Some Examples

- **Axiom:**
  \[
  \frac{A \subseteq C \quad B}{A \vdash_C B \quad A}
  \]

- **Linear arrow:**
  \[
  \begin{align*}
  & \quad \Gamma \vdash_C A \\
  & \quad \Delta, B \vdash_C C \quad L_{\rightarrow} \\
  \end{align*}
  \]
  \[
  \frac{\Gamma, A \vdash_C B}{\Gamma \vdash_C A \rightarrow B} \quad R_{\rightarrow}
  \]

- **Promotion:**
  \[
  A_1, \ldots, A_n \vdash_C B \\
  \mathcal{D}, x < p \models C \\
  x \notin \text{FV}(\mathcal{D}) \\
  p \subseteq \mathcal{D} q_i \\
  \]
  \[
  !x_{<q_1} A_1, \ldots, !x_{<q_n} A_n \vdash_{\mathcal{D}} !x_{<p} B \\
  \]

- **Existential quantification:**
  \[
  \begin{align*}
  & \quad \Gamma \vdash_C A \{p/x\} \\
  & \quad C \models \mathcal{D} \{p/x\} \quad R_{\exists x} \\
  \end{align*}
  \]
  \[
  \begin{align*}
  & \quad \Gamma, A \vdash_{\mathcal{C} \cup \mathcal{D}} C \\
  & \quad \bar{x} \notin \text{FV}(\Gamma) \cup \text{FV}(C) \cup \text{FV}(\mathcal{C}) \quad L_{\exists x} \\
  \end{align*}
  \]
  \[
  \begin{align*}
  & \quad \Gamma, \exists \bar{x} : \mathcal{D} : A \vdash_C C \\
  \end{align*}
  \]
Rules: Some Examples

- **Axiom:**
  \[
  A \subseteq C \quad B \\
  \frac{A \subseteq C}{A \vdash C} B
  \]

- **Linear arrow:**
  \[
  \frac{\Gamma \vdash C \quad \Delta, B \vdash C}{\Gamma, \Delta, A \rightarrow B \vdash C} L_\rightarrow \\
  \frac{\Gamma, A \vdash C \quad \Gamma \vdash C}{\Gamma \vdash A \rightarrow B} R_\rightarrow
  \]

- **Promotion:**
  \[
  A_1, \ldots, A_n \vdash C \quad B. \ x < p \models C \quad x \notin \text{FV}(\mathcal{D}) \quad p \subseteq \mathcal{D} q_i
  \]
  \[
  \frac{!x < q_1 A_1, \ldots, !x < q_n A_n \vdash \mathcal{D}.!x < p B}{P!}
  \]

- **Existential quantification:**
  \[
  \frac{\Gamma \vdash C \{\overline{p/x}\}}{\Gamma \vdash \exists x : \mathcal{D}. A} R_{\exists x}
  \]
  \[
  \frac{\Gamma, A \vdash C \quad \overline{x} \notin \text{FV}(\Gamma) \cup \text{FV}(C) \cup \text{FV}(\mathcal{C})}{\Gamma, \exists \overline{x} : \mathcal{D} : A \vdash C} L_{\exists x}
  \]
Rules: Some Examples

- **Axiom:**
  \[
  \frac{A \sqsubseteq C \ B}{A \vdash C \ B} A
  \]

- **Linear arrow:**
  \[
  \frac{\Gamma \vdash C \ A \ \Delta, B \vdash C \ C}{\Gamma, \Delta, A \rightarrow B \vdash C \ C} L_\rightarrow \quad \frac{\Gamma, A \vdash C \ B}{\Gamma \vdash C \ A \rightarrow B} R_\rightarrow
  \]

- **Promotion:**
  \[
  \frac{A_1, \ldots, A_n \vdash C \ B \ \mathcal{D}, x < p \models \mathcal{C} \ x \notin \text{FV}(\mathcal{D}) \ p \sqsubseteq_{\mathcal{D}} q_i}{!x < q_1 A_1, \ldots, !x < q_n A_n \vdash_{\mathcal{D}} {!x < p B} \ P!}
  \]

- **Existential quantification:**
  \[
  \frac{\Gamma \vdash \mathcal{C} \ A{p/x}}{\Gamma \vdash \mathcal{C} \{p/x\}} R_{\exists x} \quad \frac{\Gamma, A \vdash \mathcal{C} \cup \mathcal{D} \ \bar{x} \notin \text{FV}(\Gamma) \cup \text{FV}(C) \cup \text{FV}(\mathcal{C})}{\Gamma, \exists \bar{x} : \mathcal{D} : A \vdash \mathcal{C} \ C} L_{\exists x}
  \]
...is not so different from programming in BLL.

For example, the type of natural numbers remains the same:

$$N_p = \forall \alpha. !_{x < p} (\alpha(x) \rightarrow \alpha(x + 1)) \rightarrow \alpha(0) \rightarrow \alpha(p)$$

It is parametrized on a polynomial $p$.

Similarly for any arbitrary free algebra $W$.

These types support the usual impredicative iteration schema.

The added value provided by first-order quantification will show up in the embeddings.
QBAL is Still Polytime Sound.

- We prove this property by exploiting the realizability interpretation of BLL \cite{HofmannScott04}.
- Hofmann and Scott’s realizability model must be adapted to the presence of first-order quantification. Categorically:
  - **Objects** are the same.
  - The notion of a **morphism** must be slightly modified. This reflects the changes in the underlying syntax: from $\Gamma \vdash A$ in BLL to $\Gamma \vdash \mathcal{C} A$ in QBAL.

**Theorem**

*If $\pi : W_x \vdash \mathcal{C} W_p$, then the function $\llbracket \pi \rrbracket$ is computable in polynomial time.*

- **Main idea of the proof**: if $\pi$ is a proof, $\llbracket \pi \rrbracket$ is a morphism, and morphisms can be computed in polynomial time by definition.
Embedding LFPL.

- LFPL is a linear functional language [Hofmann99].
  - All functions are non-size-increasing by construction.
  - Higher-order primitive recursion.
  - Nested recursion allowed.
  - Types: $A, B ::= \Diamond | \text{Nat} | A \otimes B | A \to B$.

- The embedding:

$$
\langle \Diamond \rangle^q_p = \exists \varepsilon : \{1 \leq p\}.1
$$

$$
\langle \text{Nat} \rangle^q_p = \mathbb{N}_p
$$

$$
\langle A \otimes B \rangle^q_p = \exists (x, y) : \{x + y \leq p\}.\langle A \rangle^q_x \otimes \langle B \rangle^q_y
$$

$$
\langle A \to B \rangle^q_p = \forall (x) : \{x + p \leq q\}.\langle A \rangle^q_x \to \langle B \rangle^q_{x+p}
$$

and

$$
\langle A_1, \ldots, x : A_n \vdash M : B \downarrow \langle A_1 \rangle^q_{x_1} \sum_{i=1}^n x_i, \ldots, \langle A_n \rangle^q_{x_n} \sum_{i=1}^n x_i \vdash \emptyset \langle B \rangle^q_{\sum_{i=1}^n x_i} \rangle
$$
Embedding LFPL.

- LFPL is a linear functional language \cite{Hofmann99}.
  - All functions are non-size-increasing by construction.
  - Higher-order primitive recursion.
  - Nested recursion allowed.
  - Types: $A, B ::= \diamond | \text{Nat} | A \otimes B | A \to B$.

- The embedding:

$$
\begin{align*}
\langle \diamond \rangle_p^q &= \exists \varepsilon : \{1 \leq p\}.1 \\
\langle \text{Nat} \rangle_p^q &= \mathbb{N}_p \\
\langle A \otimes B \rangle_p^q &= \exists (x, y) : \{x + y \leq p\}.\langle A \rangle_x^q \otimes \langle B \rangle_y^q \\
\langle A \to B \rangle_p^q &= \forall (x) : \{x + p \leq q\}.\langle A \rangle_x^q \to \langle B \rangle_{x+p}^q
\end{align*}
$$

and

$$
\begin{align*}
&x : A_1, \ldots, x : A_n \vdash M : B \\
\Downarrow \\
\langle A_1 \rangle_{\sum_{i=1}^n x_i}^1, \ldots, \langle A_n \rangle_{\sum_{i=1}^n x_i}^1 \vdash \emptyset \langle B \rangle_{\sum_{i=1}^n x_i}^1
\end{align*}
$$
Embedding LFPL.

- LFPL is a linear functional language \cite{Hofmann99}.
  - All functions are non-size-increasing by construction.
  - Higher-order primitive recursion.
  - Nested recursion allowed.
- Types: $A, B ::= \Diamond \mid \text{Nat} \mid A \otimes B \mid A \rightarrow B$.

- The embedding:
  
  $\langle \Diamond \rangle^q_p = \exists \varepsilon : \{1 \leq p\}.1$
  
  $\langle \text{Nat} \rangle^q_p = \mathbb{N}_p$
  
  $\langle A \otimes B \rangle^q_p = \exists (x, y) : \{x + y \leq p\}.\langle A \rangle^q_x \otimes \langle B \rangle^q_y$
  
  $\langle A \rightarrow B \rangle^q_p = \forall (x) : \{x + p \leq q\}.\langle A \rangle^q_x \rightarrow \langle B \rangle^q_{x+p}$

and

$$x : A_1, \ldots, x : A_n \vdash M : B$$

$$\Downarrow$$

$$\langle A_1 \rangle^{\sum_{i=1}^n x_i}, \ldots, \langle A_n \rangle^{\sum_{i=1}^n x_i} \vdash \emptyset \langle B \rangle^{\sum_{i=1}^n x_i}$$
Embedding LFPL.

- LFPL is a linear functional language \cite{Hofmann99}.
  - All functions are non-size-increasing by construction.
  - Higher-order primitive recursion.
  - Nested recursion allowed.
  - Types: $A, B ::= \diamond \mid \text{Nat} \mid A \otimes B \mid A \rightarrow B$.

- The embedding:

\[
\begin{align*}
\langle \diamond \rangle_p^q &= \exists \varepsilon : \{1 \leq p\}.1 \\
\langle \text{Nat} \rangle_p^q &= \mathbb{N}_p \\
\langle A \otimes B \rangle_p^q &= \exists (x, y) : \{x + y \leq p\}.\langle A \rangle_x^q \otimes \langle B \rangle_y^q \\
\langle A \rightarrow B \rangle_p^q &= \forall (x) : \{x + p \leq q\}.\langle A \rangle_x^q \rightarrow \langle B \rangle_{x+p}^q
\end{align*}
\]

and

\[
\begin{array}{c}
x : A_1, \ldots, x : A_n \vdash M : B \\
\downarrow
\end{array}
\]

\[
\begin{array}{c}
\langle A_1 \rangle_{\sum_{i=1}^n x_i}^1, \ldots, \langle A_n \rangle_{\sum_{i=1}^n x_i}^1 \vdash \emptyset \langle B \rangle_{\sum_{i=1}^n x_i}^1
\end{array}
\]
Embedding LFPL.

- LFPL is a linear functional language [<Hofmann99>].
  - All functions are non-size-increasing by construction.
  - Higher-order primitive recursion.
  - Nested recursion allowed.
  - Types: \( A, B ::= \Diamond | \text{Nat} | A \otimes B | A \to B \).

- The embedding:

\[
\begin{align*}
\langle \Diamond \rangle_p^q &= \exists \varepsilon : \{1 \leq p\}.1 \\
\langle \text{Nat} \rangle_p^q &= \mathbb{N}_p \\
\langle A \otimes B \rangle_p^q &= \exists (x, y) : \{x + y \leq p\}.\langle A \rangle_x^q \otimes \langle B \rangle_y^q \\
\langle A \to B \rangle_p^q &= \forall (x) : \{x + p \leq q\}.\langle A \rangle_x^q \to \langle B \rangle_{x+p}^q
\end{align*}
\]

and

\[
\begin{align*}
x : A_1, \ldots, x : A_n &\vdash M : B \\
\downarrow &\\
\langle A_1 \rangle_{x_1=1}^{\sum_{i=1}^n x_i}, \ldots, \langle A_n \rangle_{x_n=1}^{\sum_{i=1}^n x_i} &\vdash \emptyset \langle B \rangle_{\sum_{i=1}^n x_i}^{\sum_{i=1}^n x_i}
\end{align*}
\]
Embedding LFPL.

- LFPL is a linear functional language \[\text{[Hofmann99]}\].
  - All functions are non-size-increasing by construction.
  - Higher-order primitive recursion.
  - Nested recursion allowed.
  - Types: \(A, B ::= \diamond \mid \text{Nat} \mid A \otimes B \mid A \rightarrow B\).

- The embedding:

\[
\begin{align*}
\langle \diamond \rangle_p^q & = \exists \varepsilon : \{1 \leq p\}.1 \\
\langle \text{Nat} \rangle_p^q & = \mathbb{N}_p \\
\langle A \otimes B \rangle_p^q & = \exists (x, y) : \{x + y \leq p\}.\langle A \rangle_x^q \otimes \langle B \rangle_y^q \\
\langle A \rightarrow B \rangle_p^q & = \forall (x) : \{x + p \leq q\}.\langle A \rangle_x^q \rightarrow \langle B \rangle_{x+p}^q
\end{align*}
\]

and

\[
\begin{align*}
x : A_1, \ldots, x : A_n \vdash M : B \\
\downarrow \\
\langle A_1 \rangle_{x_1=1}^{\sum x_i=n} x_i, \ldots, \langle A_n \rangle_{x_n=1}^{\sum x_i=n} x_i \vdash \emptyset \langle B \rangle_{\sum x_i=n}^{\sum x_i=n} x_i
\end{align*}
\]
Embedding RRW.

- RRW is a characterization of the polytime functions introduced in the nineties [Leivant93].
  - A subalgebra of the primitive recursion functions.
  - For every word algebra $W$, there are infinitely many types (tiers) $W_0, W_1, W_2, \ldots$.
  - In a recursion, the tier of the argument must be greater than the tier of the result.

- The embedding:

\[
\vdash f : W_{i_1} \times \ldots \times W_{i_n} \rightarrow W_i \\
\Downarrow \\
W_{x_1}, \ldots, W_{x_n} \vdash x_{j_1} \leq y, \ldots, x_{j_m} \leq y \quad W_{p(x_{k_1}, \ldots, x_{k_h})} + y
\]

- The proof goes by induction on the derivation for $\vdash f$.
- Interestingly, the embedding is very similar to the proof of soundness for BC, itself a close relative of RRW.
- This cannot be done in ordinary BLL.
Embedding RRW.

- RRW is a characterization of the polytime functions introduced in the nineties [Leivant93].
  - A subalgebra of the primitive recursion functions.
  - For every word algebra $W$, there are infinitely many types (tiers) $W_0, W_1, W_2, \ldots$.
  - In a recursion, the tier of the argument must be greater than the tier of the result.

- The embedding:

$$\vdash f : W_{i_1} \times \ldots \times W_{i_n} \rightarrow W_i$$

$$\Downarrow$$

$$W_{x_1}, \ldots, W_{x_n} \vdash x_{j_1} \leq y, \ldots, x_{j_m} \leq y \ W_{p(x_{k_1}, \ldots, x_{k_n})} + y$$

- The proof goes by induction on the derivation for $\vdash f$.
- Interestingly, the embedding is very similar to the proof of soundness for BC, itself a close relative of RRW.
- This cannot be done in ordinary BLL.
Embedding RRW.

- RRW is a characterization of the polytime functions introduced in the nineties [Leivant93].
  - A subalgebra of the primitive recursion functions.
  - For every word algebra $W$, there are infinitely many types (tiers) $W_0, W_1, W_2, \ldots$.
  - In a recursion, the tier of the argument must be greater than the tier of the result.

- The embedding:

$$\vdash f : W_{i_1} \times \ldots \times W_{i_n} \rightarrow W_i$$

$$\downarrow$$

$$W_{x_1}, \ldots, W_{x_n} \vdash x_{j_1} \leq y, \ldots, x_{j_m} \leq y \quad W_{p(x_{k_1}, \ldots, x_{k_h})} + y$$

- The proof goes by induction on the derivation for $\vdash f$.
- Interestingly, the embedding is very similar to the proof of soundness for BC, itself a close relative of RRW.
- This cannot be done in ordinary BLL.
Embedding RRW.

- RRW is a characterization of the polytime functions introduced in the nineties [Leivant93].
  - A subalgebra of the primitive recursion functions.
  - For every word algebra $W$, there are infinitely many types (tiers) $W_0, W_1, W_2, \ldots$.
  - In a recursion, the tier of the argument must be greater than the tier of the result.

- The embedding:

$$\vdash f : W_{i_1} \times \ldots \times W_{i_n} \rightarrow W_i$$

$$\downarrow$$

$$W_{x_1}, \ldots, W_{x_n} \vdash x_{j_1} \leq y, \ldots, x_{j_m} \leq y \quad W_{p(x_{k_1}, \ldots, x_{k_h})} + y$$

- The proof goes by induction on the derivation for $\vdash f$.
- Interestingly, the embedding is very similar to the proof of soundness for BC, itself a close relative of RRW.
- This cannot be done in ordinary BLL.
Embedding RRW.

- RRW is a characterization of the polytime functions introduced in the nineties [Leivant93].
  - A subalgebra of the primitive recursion functions.
  - For every word algebra $W$, there are infinitely many types (tiers) $W_0, W_1, W_2, \ldots$.
  - In a recursion, the tier of the argument must be greater than the tier of the result.

- The embedding:

\[
\vdash f : W_{i_1} \times \ldots \times W_{i_n} \rightarrow W_i
\]

\[
\downarrow
\]

\[
W_{x_1}, \ldots, W_{x_n} \vdash x_{j_1} \leq y, \ldots, x_{j_m} \leq y \ W_{p(x_{k_1}, \ldots, x_{k_h})} + y
\]

- The proof goes by induction on the derivation for $\vdash f$.
- Interestingly, the embedding is very similar to the proof of soundness for BC, itself a close relative of RRW.
- This cannot be done in ordinary BLL.
Embedding RRW.

- **RRW** is a characterization of the polytime functions introduced in the nineties [Leivant93].
  - A subalgebra of the primitive recursion functions.
  - For every word algebra $W$, there are infinitely many types (tiers) $W_0, W_1, W_2, \ldots$
  - In a recursion, the tier of the argument must be greater than the tier of the result.

- The embedding:

\[ \vdash f : W_{i_1} \times \ldots \times W_{i_n} \to W_i \]
\[
\downarrow
\]
\[ W_{x_1}, \ldots, W_{x_n} \vdash x_{j_1} \leq y, \ldots, x_{j_m} \leq y \ W_{p(x_{k_1}, \ldots, x_{k_h}) + y} \]

- The proof goes by induction on the derivation for $\vdash f$.
- Interestingly, the embedding is very similar to the proof of soundness for BC, itself a close relative of RRW.
- This cannot be done in ordinary BLL.
What about Light Logics?

- Here the situation is different.

\[
\begin{align*}
!A & \rightarrow 1 \\
!A & \rightarrow A \\
!A & \rightarrow !A \otimes !A \\
!A & \rightarrow !!A
\end{align*}
\]

- We were not able to embed any light logic (except SLL) into QBAL.
- Actually, ELL can be embedded into (Q)BAL.

\[
\begin{align*}
!A & \\
\downarrow & \\
!_{x<0}A
\end{align*}
\]

- But the embedding is not sensible from a dynamical point of view!
What about Light Logics?

- Here the situation is different.

\[ !A \rightarrow 1 \]
\[ !A \rightarrow A \]
\[ !A \rightarrow !A \otimes !A \]
\[ !A \rightarrow \Box !A \]

- We were not able to embed any light logic (except SLL) into QBAL.

- Actually, ELL can be embedded into (Q)BAL.

\[ !A \]
\[ \downarrow \]
\[ !x < 0 A \]

- But the embedding is not sensible from a dynamical point of view!
What about Light Logics?

- Here the situation is different.

\[ !A \rightarrow 1 \]
\[ !A \rightarrow A \]
\[ !A \rightarrow !A \otimes !A \]
\[ !A \rightarrow !!A \]

- We were not able to embed any light logic (except SLL) into QBAL.

- Actually, ELL can be embedded into (Q)BAL.

\[ !A \]
\[ \downarrow \]
\[ !x<0 A \]

- But the embedding is not sensible from a dynamical point of view!
What about Light Logics?

- Here the situation is different.

\[ !A \rightarrow 1 \]
\[ !A \rightarrow A \]
\[ !A \rightarrow !A \otimes !A \]
\[ !A \rightarrow !!A \]

- We were not able to embed any light logic (except SLL) into QBAL.

- Actually, ELL can be embedded into (Q)BAL.

\[ !A \]
\[ \downarrow \]
\[ !x<0A \]

- But the embedding is not sensible from a dynamical point of view!
What about Light Logics?

- Here the situation is different.

\[ !A \rightarrow 1 \]
\[ !A \rightarrow A \]
\[ !A \rightarrow !A \otimes !A \]
\[ !A \rightarrow !!A \]

- We were not able to embed any light logic (except SLL) into QBAL.

- Actually, ELL can be embedded into (Q)BAL.

\[ !A \]
\[ \downarrow \]
\[ !x<0A \]

- But the embedding is not sensible from a dynamical point of view!
Conclusions

- **Moral**: (a natural extension of) BLL is an interesting ICC system with strong intensional expressive power:

  ![Diagram]

  - QBAL
    - BLL
    - RRW
    - LFPL
    - SLL
    - BC

- **Price to pay**: the system is not implicit!
Conclusions

- **Moral**: (a natural extension of) BLL is an interesting ICC system with strong intensional expressive power:

  ![Diagram]

  - QBAL
    - BLL
    - RRW
    - LFPL
    - SLL
    - BC

- **Price to pay**: the system is not implicit!
Conclusions

- **Moral**: (a natural extension of) BLL is an interesting ICC system with strong intensional expressive power:

```
QBAL
/ \   /
BLL  RRW LFPL
/ \   /   /
SLL  BC  
```

- **Price to pay**: the system is not implicit!
Part II

Program Logics, Type Systems, and Relative Completeness
Floyd-Hoare Logics

- Judgments:

\[ \{P\} C \{Q\} \]

precondition \hspace{3.5cm} \text{postcondition}

\[ \text{program} \]

- Some rules:

\[ \{P[E/x]\} \ x := E \ {P} \quad \{P\} \text{skip} \ {P} \]

\[ \{P\} C \ {Q} \quad \{Q\} \ D \ {R} \]

\[ \{P\} \ C; D \ {R} \]

\[ R \Rightarrow P \quad \{P\} \ C \ {Q} \quad Q \Rightarrow S \]

\[ \{R\} \ C \ {S} \]
Floyd-Hoare Logics

- Judgments:

\[
\{P\} C \{Q\}
\]

- precondition

- postcondition

- program

- Some rules:

\[
\{P[E/x]\} \ x := E \ \{P\} \quad \{P\} \text{ skip } \{P\}
\]

\[
\frac{\{P\} \ C \ \{Q\} \quad \{Q\} \ D \ \{R\}}{\{P\} \ C; D \ \{R\}}
\]

\[
R \Rightarrow P \quad \{P\} \ C \ \{Q\} \quad Q \Rightarrow S
\]

\[
\frac{}{\{R\} \ C \ \{S\}}
\]
Relative Completeness

- The axiom system is sound.
  - If true formulas of PA are used as side-conditions.
- It’s also relatively complete [Cook78].
  - All true assertions can be derived if all true PA formulas can be used as side-conditions.
- Concrete axiom systems can be derived by throwing in a concrete sound formal system $\mathcal{F}$ for PA.
  - They are sound.
  - They are incomplete, due to Gödel incompleteness.
  - $\mathcal{F}$ is solely responsible for their incompleteness.
- A variety of FH logics enjoy the properties above.
  - Including some for higher-order programs [Honda2000]...
  - ... and some in which the complexity of programs and not only their extensional behavior is taken into account.
Program Logics

Degree of Completeness

Property Complexity

Degree of Completeness

Property Complexity
Type Systems

Degree of Completeness

Property Complexity
Type Systems
Type Systems

Degree of Completeness vs. Property Complexity

- Degree of Completeness
- Property Complexity

Diagram showing the relationship between Degree of Completeness and Property Complexity, with data points plotted along the lines.
Some Examples

- **Simply Types**
  - “Well-typed programs do not go wrong”.
  - Type inference and type checking are often decidable.

- **Dependent Types**
  - Type checking is decidable.
  - Interesting, extensional properties can be specified.

- **Intersection Types**
  - Sound and complete for termination.
  - Type inference is not decidable.
  - Studying programs as *functions* requires considering an *infinite family* of type derivations.
This Work

Degree of Completeness

Property Complexity

BLL

\( d \ell \text{PCF} \)
Why?

- dℓPCF captures both:
  - **Extensional properties of programs**: what function a program computes.
  - **Intensional properties of programs**: the time complexity of programs.

- Implicit Computational Complexity
  - Many type-theoretical characterizations of complexity classes.
  - Most of them have decidable type inference...
  - ... and poor expressive power.

- **Idea**: drop decidability constraints, and concentrate on expressivity.
  - Recover decidability by considering proper fragments.
Why?

- $d\ell$PCF captures both:
  - **Extensional properties of programs**: what function a program computes.
  - **Intensional properties of programs**: the time complexity of programs.

- **Implicit Computational Complexity**
  - Many type-theoretical characterizations of complexity classes.
  - Most of them have decidable type inference...
  - ... and poor expressive power.

- **Idea**: drop decidability constraints, and concentrate on expressivity.
  - Recover decidability by considering proper fragments.
Why?

- $d\ell PCF$ captures both:
  - **Extensional properties of programs**: what function a program computes.
  - **Intensional properties of programs**: the time complexity of programs.

- **Implicit Computational Complexity**
  - Many type-theoretical characterizations of complexity classes.
  - Most of them have decidable type inference...
  - ... and poor expressive power.

- **Idea**: drop decidability constraints, and concentrate on expressivity.
  - Recover decidability by considering proper fragments.
Why?

- $\text{dℓPCF}$ captures both:
  - **Extensional properties of programs**: what function a program computes.
  - **Intensional properties of programs**: the time complexity of programs.

- Implicit Computational Complexity
  - Many type-theoretical characterizations of complexity classes.
  - Most of them have decidable type inference...
  - ... and poor expressive power.

- **Idea**: drop decidability constraints, and concentrate on expressivity.
  - Recover decidability by considering proper fragments.
Why?

- \(d\ell\text{PCF}\) captures both:
  - **Extensional properties of programs**: what function a program computes.
  - **Intensional properties of programs**: the time complexity of programs.

- Implicit Computational Complexity
  - Many type-theoretical characterizations of complexity classes.
  - Most of them have decidable type inference...
  - ... and poor expressive power.

- **Idea**: drop decidability constraints, and concentrate on expressivity.
  - Recover decidability by considering proper fragments.
Part III

dℓPCF
A type system for the lambda calculus with constants and full higher-order recursion. (i.e. PCF).

Greatly inspired by BLL.

Indices are not necessarily polynomials, but terms from a signature $\Sigma$.

- Symbols in $\Sigma$ are given a meaning by an equational program $\mathcal{E}$.
- Side conditions in the form:

$$\phi; \Phi \models_{\mathcal{E}} I \leq J$$

Types and modal types are defined as follows:

$$\sigma, \tau ::= \text{Nat}[I, J] \mid A \to \sigma \quad \text{basic types}$$

$$A, B ::= [a < I] \cdot \sigma \quad \text{modal types}$$
A type system for the lambda calculus with constants and full higher-order recursion. (i.e. PCF).

Greatly inspired by BLL.

Indices are not necessarily polynomials, but terms from a signature $\Sigma$.

- Symbols in $\Sigma$ are given a meaning by an equational program $\mathcal{E}$.
- Side conditions in the form:

$$\phi; K_1 \leq H_1, \ldots, K_n \leq H_n \models^\mathcal{E} I \leq J$$

Types and modal types are defined as follows:

$$\sigma, \tau ::= \text{Nat}[I, J] \mid A \rightarrow \sigma$$

basic types

$$A, B ::= [a < I] \cdot \sigma$$

modal types
The Meaning of Types

\[ \left[ a < I \right] \cdot \sigma \rightarrow \tau \]

\[ \Downarrow \]

\( (\sigma\{0/a\} \otimes \ldots \otimes \sigma\{I - 1/a\}) \rightarrow \tau \)
The Meaning of Types

\[ [a < I] \cdot \sigma \rightarrow \tau \]

\[ \Downarrow \]

\[ (\sigma\{0/a\} \otimes \ldots \otimes \sigma\{I - 1/a\}) \rightarrow \tau \]
\[ \phi; \Phi \vdash^\varepsilon K \leq I \]
\[ \phi; \Phi \vdash^\varepsilon J \leq H \]
\[ \phi; \Phi \vdash^\varepsilon \text{Nat}[I, J] \subseteq \text{Nat}[K, H] \]

\[ \phi; \Phi \vdash^\varepsilon B \subseteq A \]
\[ \phi; \Phi \vdash^\varepsilon \sigma \subseteq \tau \]
\[ \phi; \Phi \vdash^\varepsilon A \rightarrow \sigma \subseteq B \rightarrow \tau \]

\[ \phi, a; \Phi, a < J \vdash^\varepsilon \sigma \subseteq \tau \]
\[ \phi; \Phi \vdash^\varepsilon J \leq I \]
\[ \phi; \Phi \vdash^\varepsilon [a < I] \cdot \sigma \subseteq [a < J] \cdot \tau \]
dℓPCF: Subtyping

\[
\begin{align*}
\phi; \Phi &\vdash^\varepsilon K \leq I \\
\phi; \Phi &\vdash^\varepsilon J \leq H \\
\hline \\
\phi; \Phi &\vdash^\varepsilon \text{Nat}[I, J] \subseteq \text{Nat}[K, H] \\
\end{align*}
\]

\[
\begin{align*}
\phi; \Phi &\vdash^\varepsilon B \subseteq A \\
\phi; \Phi &\vdash^\varepsilon \sigma \subseteq \tau \\
\hline \\
\phi; \Phi &\vdash^\varepsilon A \rightarrow \sigma \subseteq B \rightarrow \tau \\
\end{align*}
\]

\[
\begin{align*}
\phi, a; \Phi, a < J &\vdash^\varepsilon \sigma \subseteq \tau \\
\phi; \Phi &\vdash^\varepsilon J \leq I \\
\hline \\
\phi; \Phi &\vdash^\varepsilon [a < I] \cdot \sigma \subseteq [a < J] \cdot \tau \\
\end{align*}
\]
Constraints

\[
\phi; \Phi \vdash^\varepsilon [a < 1] \cdot \sigma \sqsubseteq [a < 1] \cdot \tau
\]

\[
\phi; \Phi; \Gamma, x : [a < 1] \cdot \sigma \vdash^\varepsilon_0 x : \tau \{0/a\}
\]
\(\phi; \Phi \vdash^E \text{Nat}[I + 1, J + 1] \subseteq \text{Nat}[K, H]\)

\(\phi; \Phi; \Gamma \vdash^L t : \text{Nat}[I, J]\)

\(\phi; \Phi; \Gamma \vdash^E s(t) : \text{Nat}[K, H]\)

S
\[
\phi; \Phi; \Gamma, x : [a < 1] \cdot \sigma \vdash^\xi_j t : \tau \\
\phi; \Phi; \Gamma \vdash^\xi \lambda x. t : [a < 1] \cdot \sigma \rightharpoonup \tau
\]
\[L\]
\( \phi; \Phi \vdash^\mathcal{E} \sum \subseteq \Gamma \uplus \sum_{a < I} \Delta \)

\( \phi; \Phi; \Gamma \vdash^\mathcal{E} t : [a < I] \cdot \sigma \rightarrow \tau \)

\( \phi, a; \Phi, a < I; \Delta \vdash^\mathcal{E} u : \sigma \)

\[ \frac{\phi; \Phi; \sum \vdash^\mathcal{E} J + \sum_{a \leq I} K + I \quad tu : \tau}{A} \]
\[ \phi; \Phi \vdash_\Sigma \Gamma \equiv \sum_{a < \mathcal{I}} \Delta \]
\[ \phi; \Phi; \Gamma \vdash_{\mathcal{E}} t : [a < \mathcal{I}] \cdot \sigma \rightarrow \tau \]
\[ \phi, a; \Phi, a < \mathcal{I}; \Delta \vdash_{\mathcal{E}} u : \sigma \]
\[ \phi; \Phi; \sum \vdash_{\mathcal{E}} \sum_{a \leq \mathcal{I}} \sum_{a \in \mathcal{I}} t u : \tau \]
dℓPCF: Some Rules

Bounded Sum of Modal Types

\[ \phi; \Phi \vdash_{\mathcal{E}} \Sigma \equiv \Gamma \uplus \sum_{a < I} \Delta \]

\[ \phi; \Phi, \Gamma \vdash_{\mathcal{E}} t : [a < I] \cdot \sigma \rightarrow \tau \]

\[ \phi, a; \Phi, a < I; \Delta \vdash_{\mathcal{E}} u : \sigma \]

\[ \phi; \Phi; \Sigma \vdash_{\mathcal{E}} \sum_{a < I}^{K+I} tu : \tau \]

A
$a; \emptyset; \emptyset \vdash_1 t : [b < J] \cdot \text{Nat}[a] \rightarrow \text{Nat}[K]$

What does this mean?

- $t$ computes a function from natural numbers to natural numbers.
- Something extensional:
  - On input a natural number $n$, $t$ returns a natural number $K(n/a)$
- Something more intensional:
  - The cost of evaluation of $t$ on an input $n$ is $(I + J)(n/a)$.
- Two questions:
  - Is this correct?
  - How many programs can be captured this way?
\[ a; \emptyset; \emptyset \vdash_I t : [b < J] \cdot \text{Nat}[a] \rightarrow \text{Nat}[K] \]

What does this mean?

- \( t \) computes a function from natural numbers to natural numbers.
- Something **extensional**:  
  - On input a natural number \( n \), \( t \) returns a natural number \( K\{n/a\} \)
- Something more **intensional**:  
  - The cost of evaluation of \( t \) on an input \( n \) is \( (I + J)\{n/a\} \).

- Two questions:  
  - Is this **correct**?  
  - How many programs can be captured this way?
\[
a; \emptyset; \emptyset \vdash t : [b < J] \cdot \text{Nat}[a] \rightarrow \text{Nat}[K]
\]

What does this mean?

- \(t\) computes a function from natural numbers to natural numbers.
- Something **extensional**:
  - On input a natural number \(n\), \(t\) returns a natural number \(K\{n/a\}\)
- Something more **intensional**:
  - The cost of evaluation of \(t\) on an input \(n\) is \((I + J)\{n/a\}\).

- Two questions:
  - Is this correct?
  - How many programs can be captured this way?
\[ a; \emptyset; \emptyset \vdash t : [b < J] \cdot \text{Nat}[a] \rightarrow \text{Nat}[K] \]

**What does this mean?**

- \( t \) computes a function from natural numbers to natural numbers.
- Something **extensional**:
  - On input a natural number \( n \), \( t \) returns a natural number \( K\{n/a\} \)
- Something more **intensional**:
  - The cost of evaluation of \( t \) on an input \( n \) is \( (I + J)\{n/a\} \).
- Two questions:
  - Is this **correct**?
  - How many programs can be captured this way?
Intensional Soundness

- A generalization of KAM which takes constants and fixpoints into account.
- Lift the type system to closures, stack and environments.

Lemma (Measure Decreasing)

Suppose \((t, \epsilon, \epsilon) \rightarrow^* D \rightarrow E\) and let \(D\) have weight \(I\). Then one of the following holds:

1. \(E\) has weight \(J\), \(\phi; \Phi \models I = J\) but \(|D| > |E|\);
2. \(E\) has weight \(J\), \(\phi; \Phi \models I > J\) and \(|E| < |D| + |t|\);

Theorem

Let \(\emptyset; \emptyset; \emptyset \vdash_I t : \text{Nat}[J, K]\) and \(t \downarrow^n m\). Then \(n \leq |t| \cdot [I]_\rho^\epsilon\)
Intensional Soundness

- A generalization of KAM which takes constants and fixpoints into account.
- Lift the type system to closures, stack and environments.

Lemma (Measure Decreasing)

Suppose \((t, \epsilon, \epsilon) \rightarrow^* D \rightarrow E\) and let \(D\) have weight \(I\). Then one of the following holds:
1. \(E\) has weight \(J\), \(\phi; \Phi \models I = J\) but \(|D| > |E|\);
2. \(E\) has weight \(J\), \(\phi; \Phi \models I > J\) and \(|E| < |D| + |t|\);

Theorem

Let \(\emptyset; \emptyset; \emptyset \vdash_I t : \text{Nat}[J, K]\) and \(t \downarrow^n m\). Then \(n \leq |t| \cdot [I]_\rho^E\)
Completeness for Programs

- The following holds only when $\mathcal{E}$ is universal.
- $(|\sigma|)$ is the PCF type underlying $\sigma$, i.e. its skeleton.

**Lemma (Weighted Subject Expansion)**

If $D$ has weight $I$ and type $\sigma$ and $C$ is typable with type $(|\sigma|)$. Then, $C \rightarrow D$ implies that $C$ has weight $J$ and type $\sigma$, where $\phi; \Phi \models J \leq I + 1$.

**Theorem (Relative Completeness for Programs)**

Let $t$ be a PCF program such that $t \Downarrow^n m$. Then, there exist two index terms $I$ and $J$ such that $[I]^U \leq n$ and $[J]^U = m$ and such that the term $t$ is typable in d\textit{l}PCF as $\emptyset; \emptyset; \emptyset \vdash^U t : \text{Nat}[J]$. 
Completeness for Programs

- The following holds only when $\mathcal{E}$ is universal.
- $(|\sigma|)$ is the PCF type underlying $\sigma$, i.e. its skeleton.

Lemma (Weighted Subject Expansion)

If $D$ has weight $I$ and type $\sigma$ and $C$ is typable with type $(|\sigma|)$. Then, $C \to D$ implies that $C$ has weight $J$ and type $\sigma$, where $\phi; \Phi \models J \leq I + 1$.

Theorem (Relative Completeness for Programs)

Let $t$ be a PCF program such that $t \downarrow^n m$. Then, there exist two index terms $I$ and $J$ such that $[I]^U \leq n$ and $[J]^U = m$ and such that the term $t$ is typable in dℓPCF as $\emptyset; \emptyset; \emptyset \vdash^U t : \text{Nat}[J]$. 

It strongly relies on the universality of $\mathcal{U}$.

Suppose that $\{\pi_n\}_{n \in \mathbb{N}}$ is an r.e. family of type derivations:

- For the same term $t$;
- Having the same PCF skeleton (as type derivations);

Then we can turn them into a single, parametric type derivation.

**Theorem (Relative Completeness for Functions)**

Suppose that $t$ is a PCF term such that $\vdash t : \text{Nat} \rightarrow \text{Nat}$.
Moreover, suppose that there are two (total and computable) functions $f, g : \mathbb{N} \rightarrow \mathbb{N}$ such that $t \upharpoonright^{g(n)} f(n)$. Then there are terms $I, J, K$ with $[I + J] \leq g$ and $[K] = f$, such that

$$a; \emptyset; \emptyset \vdash^\mathcal{U}_I t : [b < J] \cdot \text{Nat}[a] \rightarrow \text{Nat}[K].$$
Completeness for Functions

- It strongly relies on the universality of $\mathcal{U}$.
- Suppose that $\{\pi_n\}_{n \in \mathbb{N}}$ is an r.e. family of type derivations:
  - For the same term $t$;
  - Having the same PCF skeleton (as type derivations);
Then we can turn them into a single, parametric type derivation.

**Theorem (Relative Completeness for Functions)**

*Suppose that $t$ is a PCF term such that $\vdash t : \text{Nat} \rightarrow \text{Nat}$. Moreover, suppose that there are two (total and computable) functions $f, g : \mathbb{N} \rightarrow \mathbb{N}$ such that $t \downarrow^{g(n)} f(n)$. Then there are terms $I, J, K$ with $[I + J] \leq g$ and $[K] = f$, such that*

$$a; \emptyset; \emptyset \vdash^\mathcal{U} t : [b < J] \cdot \text{Nat}[a] \rightarrow \text{Nat}[K].$$
Conclusions

- A relatively complete type system $dlPCF$.
- Type inference, type checking and derivation checking are undecidable, in general.
  - ... but can become manageable if $E$ is simple enough.
  - Light Logics!
- Another result: relative decidability of type inference.
Thank you!

Questions?