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 $\mathcal{S}$ is **intensionally** more powerful than another one, $\mathcal{P}$?
- One possibility is the one advocated by Felleisen *[Felleisen95]*:
  - $\mathcal{S}$ is (strictly) more expressive than $\mathcal{P}$ iff $\mathcal{S}$ is a syntactical extension of $\mathcal{P}$ and the constructs characterizing $\mathcal{S}$ 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 $\mathcal{S}$ is **intensionally** more powerful than another one, $\mathcal{P}$?
- One possibility is the one advocated by Felleisen *[Felleisen95]*:
  - $\mathcal{S}$ is (strictly) more expressive than $\mathcal{P}$ iff $\mathcal{S}$ is a syntactical extension of $\mathcal{P}$ and the constructs characterizing $\mathcal{S}$ 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 $\mathcal{S}$ is **intensionally** more powerful than another one, $\mathcal{P}$?
- One possibility is the one advocated by Felleisen *[Felleisen95]*:
  - $\mathcal{S}$ is (strictly) more expressive than $\mathcal{P}$ iff $\mathcal{S}$ is a syntactical extension of $\mathcal{P}$ and the constructs characterizing $\mathcal{S}$ 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 $\mathcal{S}$ is **intensionally** more powerful than another one, $\mathcal{P}$?
- One possibility is the one advocated by Felleisen *[Felleisen95]*:
  - $\mathcal{S}$ is (strictly) more expressive than $\mathcal{P}$ iff $\mathcal{S}$ is a syntactical extension of $\mathcal{P}$ and the constructs characterizing $\mathcal{S}$ 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 $\mathcal{S}$ is **intensionally** more powerful than another one, $\mathcal{P}$?
- One possibility is the one advocated by Felleisen *[Felleisen95]*:
  - $\mathcal{S}$ is (strictly) more expressive than $\mathcal{P}$ iff $\mathcal{S}$ is a syntactical extension of $\mathcal{P}$ and the constructs characterizing $\mathcal{S}$ 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.

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.

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:

$$!_n A \multimap 1$$
$$!_{1+n} A \multimap A$$
$$!_{n+m} A \multimap !_n A \otimes !_m A$$
$$!_{nm} A \multimap !_n !_m A$$

  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:

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

- 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:

$$!_n A \multimap 1$$
$$!_{1+n} A \multimap A$$
$$!_{n+m} A \multimap !_n A \otimes !_m A$$
$$!_{nm} A \multimap !_n !_m A$$

  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:

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

- 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:

$$!_n A \multimap 1$$
$$!_{1+n} A \multimap A$$
$$!_{n+m} A \multimap !_n A \otimes !_m A$$
$$!_{nm} A \multimap !_n !_m A$$

  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:

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

- Intuitively:

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

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:

$$\exists(\overline{x}) : \mathscr{C}.A$$
$$\forall(\overline{x}) : \mathscr{C}.A$$

  where $\mathscr{C}$ is a set of constraints.
- Example:

$$\exists(x, y) : \{x \leqslant z^2, y \leqslant x\}.!_{xy^2} A$$

  Notice that the constraints in $\{x \leqslant z^2, y \leqslant x\}$ enforce a polynomial upper bound on both x and y: $x \leqslant z^2$ and $y \leqslant x \leqslant z^2$.
  - Not by coincidence!
- Sequents have the form $\Gamma \vdash_{\mathscr{C}} A$.
- Rules for first-order quantifier are standard.
- The rules coming from BLL leaves $\mathscr{C}$ unchanged.

Rules: Some Examples

- Axiom:

$$\frac{A \sqsubseteq_{\mathscr{C}} B}{A \vdash_{\mathscr{C}} B}\ A$$

- Linear arrow:

$$\frac{\Gamma \vdash_{\mathscr{C}} A \quad \Delta, B \vdash_{\mathscr{C}} C}{\Gamma, \Delta, A \multimap B \vdash_{\mathscr{C}} C}\ L_{\multimap} \qquad \frac{\Gamma, A \vdash_{\mathscr{C}} B}{\Gamma \vdash_{\mathscr{C}} A \multimap B}\ R_{\multimap}$$

- Promotion:

$$\frac{A_1, \ldots, A_n \vdash_{\mathscr{C}} B \quad \mathscr{D}, x < p \models \mathscr{C} \quad x \notin FV(\mathscr{D}) \quad p \sqsubseteq_{\mathscr{D}} q_i}{!_{x<q_1} A_1, \ldots, !_{x<q_n} A_n \vdash_{\mathscr{D}} !_{x<p} B}\ P_!$$

- Existential quantification:

$$\frac{\Gamma \vdash_{\mathscr{C}} A\{\overline{p}/\overline{x}\} \quad \mathscr{C} \models \mathscr{D}\{\overline{p}/\overline{x}\}}{\Gamma \vdash_{\mathscr{C}} \exists \overline{x} : \mathscr{D}.A}\ R_{\exists x}$$

$$\frac{\Gamma, A \vdash_{\mathscr{C} \cup \mathscr{D}} C \quad \overline{x} \notin FV(\Gamma) \cup FV(C) \cup FV(\mathscr{C})}{\Gamma, \exists \overline{x} : \mathscr{D} : A \vdash_{\mathscr{C}} C}\ L_{\exists x}$$

Rules: Some Examples

- Axiom:

$$\frac{A \sqsubseteq_{\mathscr{C}} B}{A \vdash_{\mathscr{C}} B}\ A$$

- Linear arrow:

$$\frac{\Gamma \vdash_{\mathscr{C}} A \quad \Delta, B \vdash_{\mathscr{C}} C}{\Gamma, \Delta, A \multimap B \vdash_{\mathscr{C}} C}\ L_{\multimap} \qquad \frac{\Gamma, A \vdash_{\mathscr{C}} B}{\Gamma \vdash_{\mathscr{C}} A \multimap B}\ R_{\multimap}$$

- Promotion:

$$\frac{A_1, \ldots, A_n \vdash_{\mathscr{C}} B \quad \mathscr{D}, x < p \models \mathscr{C} \quad x \notin FV(\mathscr{D}) \quad p \sqsubseteq_{\mathscr{D}} q_i}{!_{x<q_1} A_1, \ldots, !_{x<q_n} A_n \vdash_{\mathscr{D}} !_{x<p} B}\ P_!$$

- Existential quantification:

$$\frac{\Gamma \vdash_{\mathscr{C}} A\{\overline{p}/\overline{x}\} \quad \mathscr{C} \models \mathscr{D}\{\overline{p}/\overline{x}\}}{\Gamma \vdash_{\mathscr{C}} \exists \overline{x} : \mathscr{D}.A}\ R_{\exists x}$$

$$\frac{\Gamma, A \vdash_{\mathscr{C} \cup \mathscr{D}} C \quad \overline{x} \notin FV(\Gamma) \cup FV(C) \cup FV(\mathscr{C})}{\Gamma, \exists \overline{x} : \mathscr{D} : A \vdash_{\mathscr{C}} C}\ L_{\exists x}$$

Rules: Some Examples

- Axiom:

$$\frac{A \sqsubseteq_{\mathscr{C}} B}{A \vdash_{\mathscr{C}} B}\ A$$

- Linear arrow:

$$\frac{\Gamma \vdash_{\mathscr{C}} A \quad \Delta, B \vdash_{\mathscr{C}} C}{\Gamma, \Delta, A \multimap B \vdash_{\mathscr{C}} C}\ L_{\multimap} \qquad \frac{\Gamma, A \vdash_{\mathscr{C}} B}{\Gamma \vdash_{\mathscr{C}} A \multimap B}\ R_{\multimap}$$

- Promotion:

$$\frac{A_1, \ldots, A_n \vdash_{\mathscr{C}} B \quad \mathscr{D}, x < p \models \mathscr{C} \quad x \notin FV(\mathscr{D}) \quad p \sqsubseteq_{\mathscr{D}} q_i}{!_{x<q_1} A_1, \ldots, !_{x<q_n} A_n \vdash_{\mathscr{D}} !_{x<p} B}\ P_!$$

- Existential quantification:

$$\frac{\Gamma \vdash_{\mathscr{C}} A\{\overline{p}/\overline{x}\} \quad \mathscr{C} \models \mathscr{D}\{\overline{p}/\overline{x}\}}{\Gamma \vdash_{\mathscr{C}} \exists \overline{x} : \mathscr{D}.A}\ R_{\exists x}$$

$$\frac{\Gamma, A \vdash_{\mathscr{C} \cup \mathscr{D}} C \quad \overline{x} \notin FV(\Gamma) \cup FV(C) \cup FV(\mathscr{C})}{\Gamma, \exists \overline{x} : \mathscr{D} : A \vdash_{\mathscr{C}} C}\ L_{\exists x}$$

Rules: Some Examples

- Axiom:

$$\frac{A \sqsubseteq_{\mathscr{C}} B}{A \vdash_{\mathscr{C}} B}\ A$$

- Linear arrow:

$$\frac{\Gamma \vdash_{\mathscr{C}} A \quad \Delta, B \vdash_{\mathscr{C}} C}{\Gamma, \Delta, A \multimap B \vdash_{\mathscr{C}} C}\ L_{\multimap} \qquad \frac{\Gamma, A \vdash_{\mathscr{C}} B}{\Gamma \vdash_{\mathscr{C}} A \multimap B}\ R_{\multimap}$$

- Promotion:

$$\frac{A_1, \ldots, A_n \vdash_{\mathscr{C}} B \quad \mathscr{D}, x < p \models \mathscr{C} \quad x \notin FV(\mathscr{D}) \quad p \sqsubseteq_{\mathscr{D}} q_i}{!_{x<q_1} A_1, \ldots, !_{x<q_n} A_n \vdash_{\mathscr{D}} !_{x<p} B}\ P_!$$

- Existential quantification:

$$\frac{\Gamma \vdash_{\mathscr{C}} A\{\overline{p}/\overline{x}\} \quad \mathscr{C} \models \mathscr{D}\{\overline{p}/\overline{x}\}}{\Gamma \vdash_{\mathscr{C}} \exists \overline{x} : \mathscr{D}.A}\ R_{\exists x}$$

$$\frac{\Gamma, A \vdash_{\mathscr{C} \cup \mathscr{D}} C \quad \overline{x} \notin FV(\Gamma) \cup FV(C) \cup FV(\mathscr{C})}{\Gamma, \exists \overline{x} : \mathscr{D} : A \vdash_{\mathscr{C}} C}\ L_{\exists x}$$

Programming in QBAL...

- ...is not so different from programming in BLL.
- For example, the type of natural numbers remains the same:

$$\mathbf{N}_p = \forall\alpha.!_{x<p}(\alpha(x) \multimap \alpha(x+1)) \multimap \alpha(0) \multimap \alpha(p)$$

  It is parametrized on a polynomial p.
- Similarly for any arbitrary free algebra $\mathbf{W}$.
- 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 *[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 $\Gamma \vdash_{\mathscr{C}} A$ in QBAL.

Theorem

If $\pi : \mathbf{W}_x \vdash_{\varnothing} \mathbf{W}_p$, *then the function* $[\![\pi]\!]$ *is computable in polynomial time.*

- **Main idea of the proof**: if π is a proof, $[\![\pi]\!]$ 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: $A, B ::= \diamond \mid Nat \mid A \otimes B \mid A \multimap B$.
- The embedding:

$$\langle \diamond \rangle_p^q = \exists \varepsilon : \{1 \leqslant p\}.1$$
$$\langle Nat \rangle_p^q = \mathbf{N}_p$$
$$\langle A \otimes B \rangle_p^q = \exists (x, y) : \{x + y \leqslant p\}.\langle A \rangle_x^q \otimes \langle B \rangle_y^q$$
$$\langle A \multimap B \rangle_p^q = \forall (x) : \{x + p \leqslant q\}.\langle A \rangle_x^q \multimap \langle B \rangle_{x+p}^q$$

and

$$x : A_1, \ldots, x : A_n \vdash M : B$$
$$\Downarrow$$
$$\langle A_1 \rangle_{x_1}^{\sum_{i=1}^n x_i}, \ldots, \langle A_n \rangle_{x_n}^{\sum_{i=1}^n x_i} \vdash_\varnothing \langle B \rangle_{\sum_{i=1}^n x_i}^{\sum_{i=1}^n x_i}$$

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: $A, B ::= \diamond \mid Nat \mid A \otimes B \mid A \multimap B$.
- The embedding:

$$\langle \diamond \rangle_p^q = \exists \varepsilon : \{1 \leqslant p\}.1$$
$$\langle Nat \rangle_p^q = \mathbf{N}_p$$
$$\langle A \otimes B \rangle_p^q = \exists (x, y) : \{x + y \leqslant p\}.\langle A \rangle_x^q \otimes \langle B \rangle_y^q$$
$$\langle A \multimap B \rangle_p^q = \forall (x) : \{x + p \leqslant q\}.\langle A \rangle_x^q \multimap \langle B \rangle_{x+p}^q$$

and

$$x : A_1, \ldots, x : A_n \vdash M : B$$
$$\Downarrow$$
$$\langle A_1 \rangle_{x_1}^{\sum_{i=1}^n x_i}, \ldots, \langle A_n \rangle_{x_n}^{\sum_{i=1}^n x_i} \vdash_{\varnothing} \langle B \rangle_{\sum_{i=1}^n x_i}^{\sum_{i=1}^n x_i}$$

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: $A, B ::= \diamond \mid Nat \mid A \otimes B \mid A \multimap B$.
- The embedding:

$$\begin{aligned}
\langle \diamond \rangle_p^q &= \exists \varepsilon : \{1 \leqslant p\}.1 \\
\langle Nat \rangle_p^q &= \mathbf{N}_p \\
\langle A \otimes B \rangle_p^q &= \exists (x, y) : \{x + y \leqslant p\}.\langle A \rangle_x^q \otimes \langle B \rangle_y^q \\
\langle A \multimap B \rangle_p^q &= \forall (x) : \{x + p \leqslant q\}.\langle A \rangle_x^q \multimap \langle B \rangle_{x+p}^q
\end{aligned}$$

and

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

$$\Downarrow$$

$$\langle A_1 \rangle_{x_1}^{\sum_{i=1}^n x_i}, \ldots, \langle A_n \rangle_{x_n}^{\sum_{i=1}^n x_i} \vdash_{\varnothing} \langle B \rangle_{\sum_{i=1}^n x_i}^{\sum_{i=1}^n x_i}$$

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: $A, B ::= \diamond \mid Nat \mid A \otimes B \mid A \multimap B$.
- The embedding:

$$\begin{aligned}
\langle \diamond \rangle_p^q &= \exists \varepsilon : \{1 \leqslant p\}.1 \\
\langle Nat \rangle_p^q &= \mathbf{N}_p \\
\langle A \otimes B \rangle_p^q &= \exists (x, y) : \{x + y \leqslant p\}.\langle A \rangle_x^q \otimes \langle B \rangle_y^q \\
\langle A \multimap B \rangle_p^q &= \forall (x) : \{x + p \leqslant q\}.\langle A \rangle_x^q \multimap \langle B \rangle_{x+p}^q
\end{aligned}$$

and

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

$$\Downarrow$$

$$\langle A_1 \rangle_{x_1}^{\sum_{i=1}^n x_i}, \ldots, \langle A_n \rangle_{x_n}^{\sum_{i=1}^n x_i} \vdash_{\varnothing} \langle B \rangle_{\sum_{i=1}^n x_i}^{\sum_{i=1}^n x_i}$$

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: $A, B ::= \diamond \mid Nat \mid A \otimes B \mid A \multimap B$.
- The embedding:

$$\begin{aligned}
\langle \diamond \rangle_p^q &= \exists \varepsilon : \{1 \leqslant p\}.1 \\
\langle Nat \rangle_p^q &= \mathbf{N}_p \\
\langle A \otimes B \rangle_p^q &= \exists (x, y) : \{x + y \leqslant p\}.\langle A \rangle_x^q \otimes \langle B \rangle_y^q \\
\langle A \multimap B \rangle_p^q &= \forall (x) : \{x + p \leqslant q\}.\langle A \rangle_x^q \multimap \langle B \rangle_{x+p}^q
\end{aligned}$$

and

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

$$\Downarrow$$

$$\langle A_1 \rangle_{x_1}^{\sum_{i=1}^n x_i}, \ldots, \langle A_n \rangle_{x_n}^{\sum_{i=1}^n x_i} \vdash_{\varnothing} \langle B \rangle_{\sum_{i=1}^n x_i}^{\sum_{i=1}^n x_i}$$

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: $A, B ::= \diamond \mid Nat \mid A \otimes B \mid A \multimap B$.
- The embedding:

$$\begin{aligned}
\langle \diamond \rangle_p^q &= \exists \varepsilon : \{1 \leqslant p\}.1 \\
\langle Nat \rangle_p^q &= \mathbf{N}_p \\
\langle A \otimes B \rangle_p^q &= \exists (x, y) : \{x + y \leqslant p\}.\langle A \rangle_x^q \otimes \langle B \rangle_y^q \\
\langle A \multimap B \rangle_p^q &= \forall (x) : \{x + p \leqslant q\}.\langle A \rangle_x^q \multimap \langle B \rangle_{x+p}^q
\end{aligned}$$

and

$$\begin{array}{c}
x : A_1, \ldots, x : A_n \vdash M : B \\
\Downarrow \\
\langle A_1 \rangle_{x_1}^{\sum_{i=1}^n x_i}, \ldots, \langle A_n \rangle_{x_n}^{\sum_{i=1}^n x_i} \vdash_{\varnothing} \langle B \rangle_{\sum_{i=1}^n x_i}^{\sum_{i=1}^n x_i}
\end{array}$$

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:

$$\vdash f : W_{i_1} \times \ldots \times W_{i_n} \to W_i$$
$$\Downarrow$$
$$\mathbf{W}_{x_1}, \ldots, \mathbf{W}_{x_n} \vdash_{x_{j_1} \leqslant y, \ldots, x_{j_m} \leqslant y} \mathbf{W}_{p(x_{k_1}, \ldots, x_{k_h}) + y}$$

- 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:

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

$$\Downarrow$$

$$\mathbf{W}_{x_1}, \ldots, \mathbf{W}_{x_n} \vdash_{x_{j_1} \leqslant y, \ldots, x_{j_m} \leqslant y} \mathbf{W}_{p(x_{k_1}, \ldots, x_{k_h}) + y}$$

- 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:

$$\vdash f : W_{i_1} \times \ldots \times W_{i_n} \to W_i$$
$$\Downarrow$$
$$\mathbf{W}_{x_1}, \ldots, \mathbf{W}_{x_n} \vdash_{x_{j_1} \leqslant y, \ldots, x_{j_m} \leqslant y} \mathbf{W}_{p(x_{k_1}, \ldots, x_{k_h}) + y}$$

- 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:

$$\vdash f : W_{i_1} \times \ldots \times W_{i_n} \to W_i$$
$$\Downarrow$$
$$\mathbf{W}_{x_1}, \ldots, \mathbf{W}_{x_n} \vdash_{x_{j_1} \leqslant y, \ldots, x_{j_m} \leqslant y} \mathbf{W}_{p(x_{k_1}, \ldots, x_{k_h}) + y}$$

- 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:

$$\vdash f : W_{i_1} \times \ldots \times W_{i_n} \to W_i$$
$$\Downarrow$$
$$\mathbf{W}_{x_1}, \ldots, \mathbf{W}_{x_n} \vdash_{x_{j_1} \leqslant y, \ldots, x_{j_m} \leqslant y} \mathbf{W}_{p(x_{k_1}, \ldots, x_{k_h}) + y}$$

- 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:

$$\vdash f : W_{i_1} \times \ldots \times W_{i_n} \to W_i$$
$$\Downarrow$$
$$\mathbf{W}_{x_1}, \ldots, \mathbf{W}_{x_n} \vdash_{x_{j_1} \leqslant y, \ldots, x_{j_m} \leqslant y} \mathbf{W}_{p(x_{k_1}, \ldots, x_{k_h})+y}$$

- 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.

$$!A \multimap 1$$
$$!A \multimap A$$
$$!A \multimap !A \otimes !A$$
$$!A \multimap !!A$$

- 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.

$$!A \multimap 1$$
$$!A \multimap A$$
$$!A \multimap !A \otimes !A$$
$$!A \multimap !!A$$

- 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.

$$!A \multimap 1$$
$$!A \multimap A$$
$$!A \multimap !A \otimes !A$$
$$!A \multimap !!A$$

- 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.

$$!A \multimap 1$$
$$!A \multimap A$$
$$!A \multimap !A \otimes !A$$
$$!A \multimap !!A$$

- 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.

$$!A \multimap 1$$
$$!A \multimap A$$
$$!A \multimap !A \otimes !A$$
$$!A \multimap !!A$$

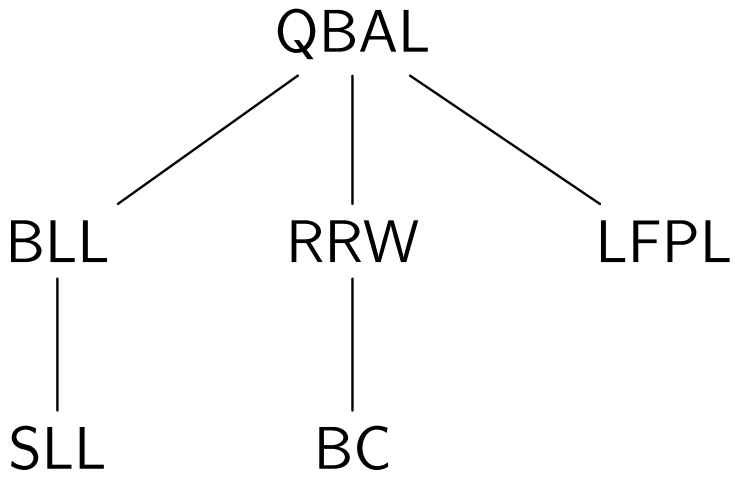
- We were not able to embed any light logic (except SLL) into QBAL.
- Actually, ELL can be embedded into (Q)BAL.

$$!A$$
$$\Downarrow$$
$$!_{x<0} A$$

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

Conclusions

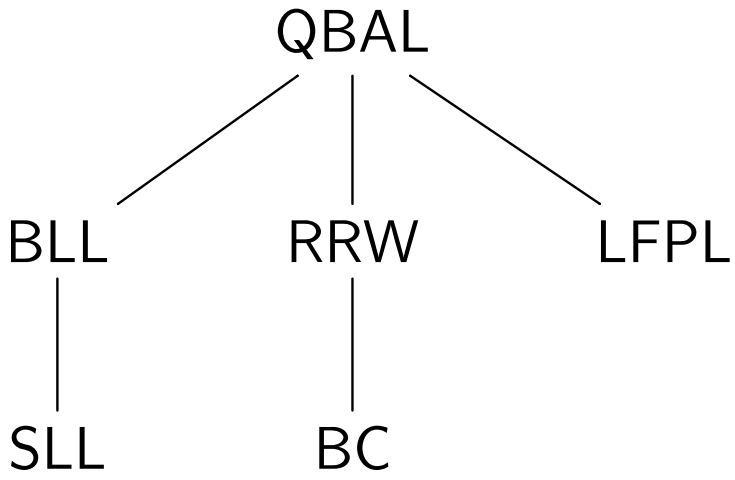
- **Moral**: (a natural extension of) BLL is an interesting ICC system with strong intensional expressive power:



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

Conclusions

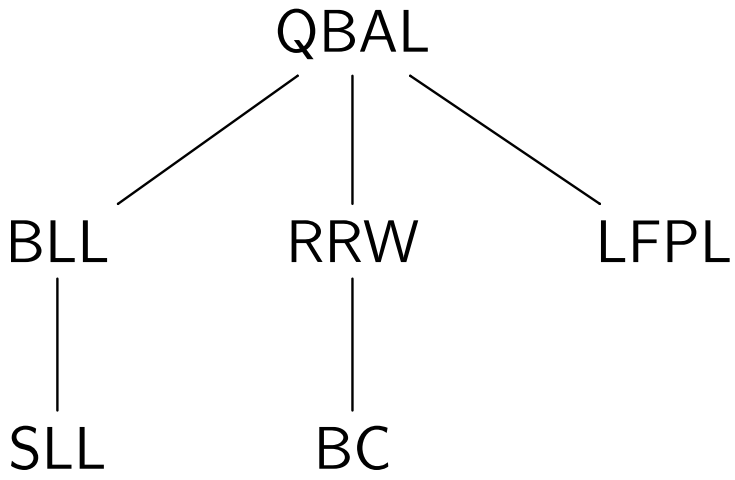
- **Moral**: (a natural extension of) BLL is an interesting ICC system with strong intensional expressive power:



- **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:



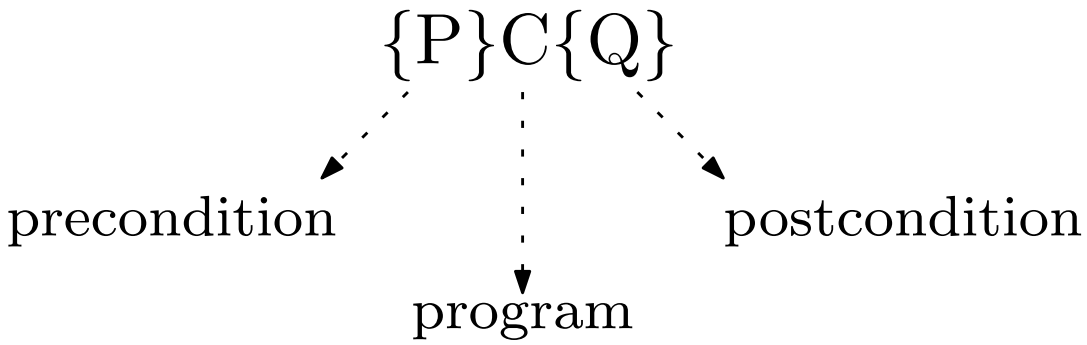
- **Price to pay**: the system is not implicit!

Part II

Program Logics, Type Systems, and Relative Completeness

Floyd-Hoare Logics

- Judgments:



- Some rules:

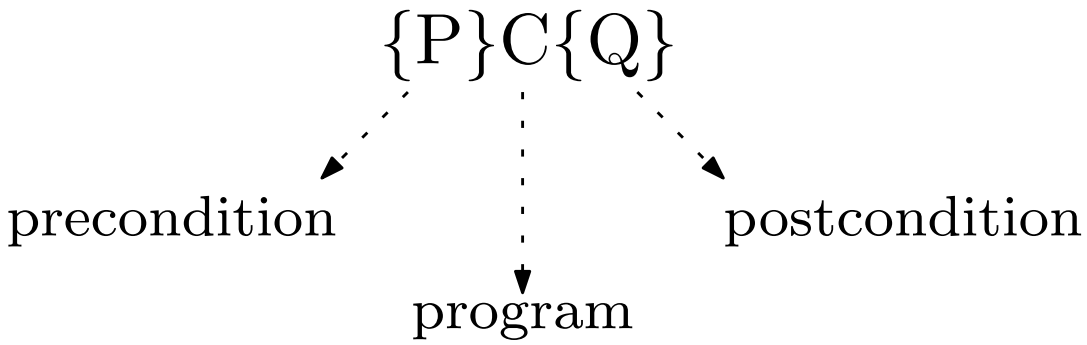
$$\frac{}{\{P[E/x]\}\ x := E\ \{P\}} \qquad \frac{}{\{P\}\ \textbf{skip}\ \{P\}}$$

$$\frac{\{P\}\ C\ \{Q\} \quad \{Q\}\ D\ \{R\}}{\{P\}\ C; D\ \{R\}}$$

$$\frac{R \Rightarrow P \quad \{P\}\ C\ \{Q\} \quad Q \Rightarrow S}{\{R\}\ C\ \{S\}}$$

Floyd-Hoare Logics

- Judgments:



- Some rules:

$$\frac{}{\{P[E/x]\}\ x := E\ \{P\}} \qquad \frac{}{\{P\}\ \textbf{skip}\ \{P\}}$$

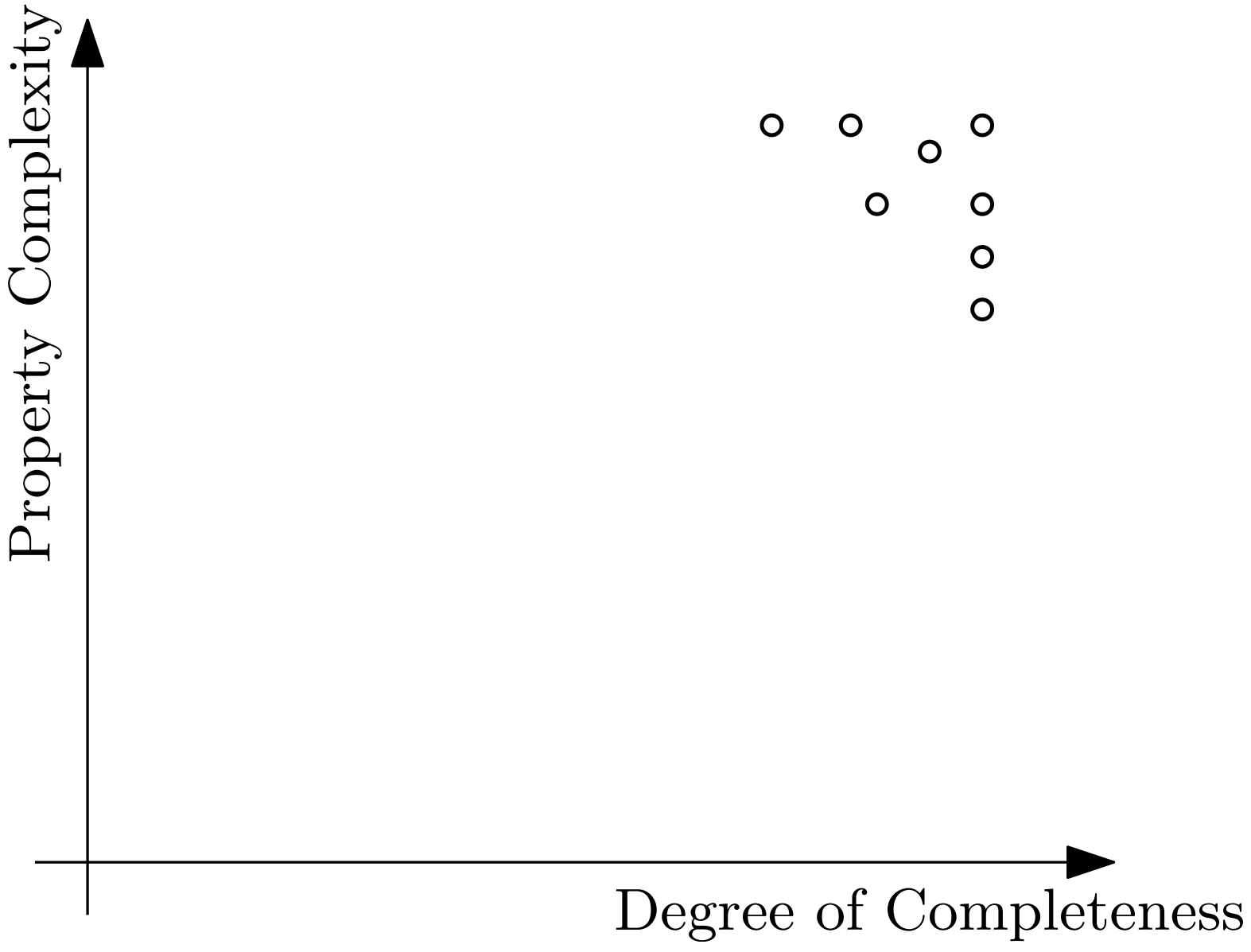
$$\frac{\{P\}\ C\ \{Q\} \quad \{Q\}\ D\ \{R\}}{\{P\}\ C; D\ \{R\}}$$

$$\frac{R \Rightarrow P \quad \{P\}\ C\ \{Q\} \quad Q \Rightarrow S}{\{R\}\ C\ \{S\}}$$

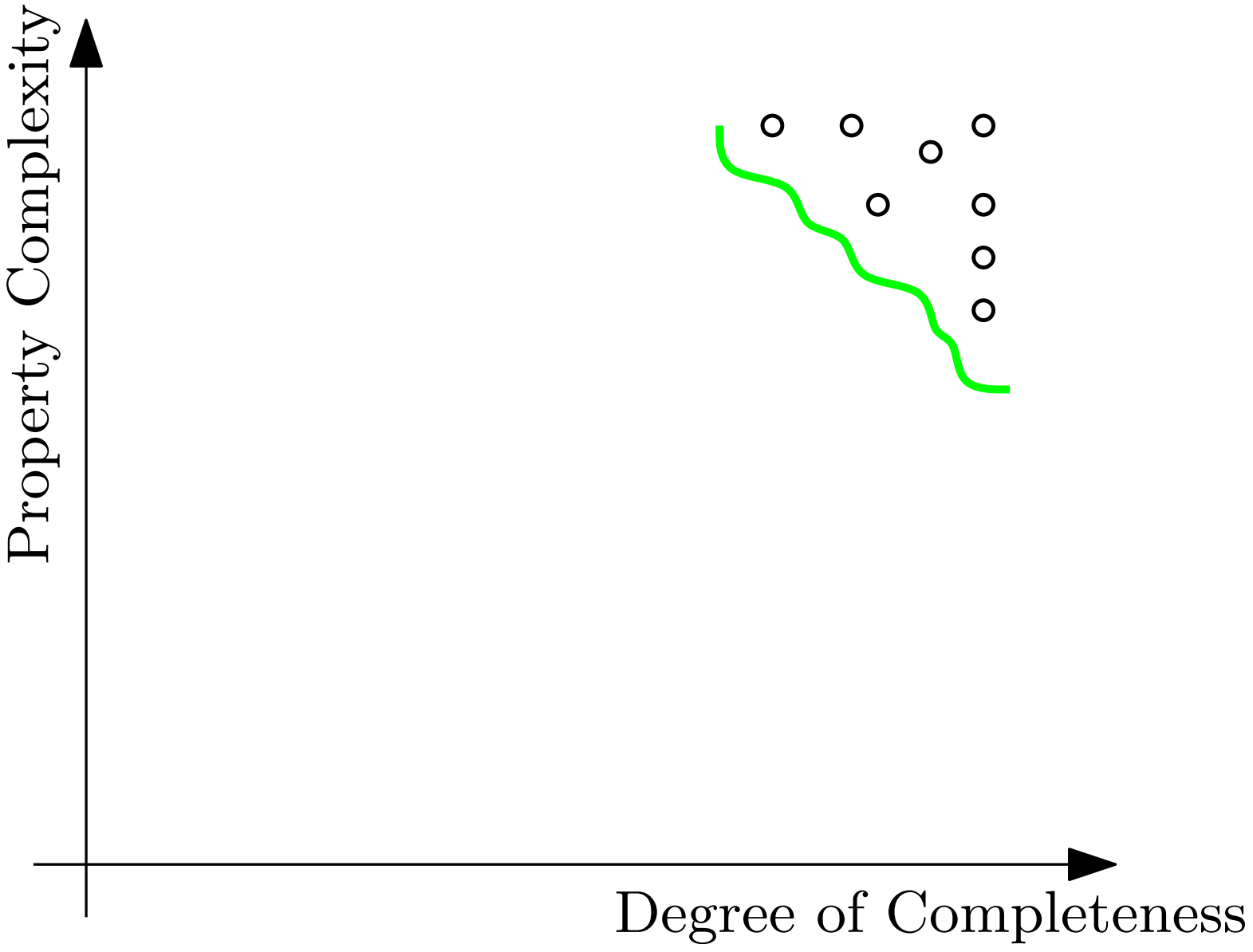
Relative Completeness

- 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.

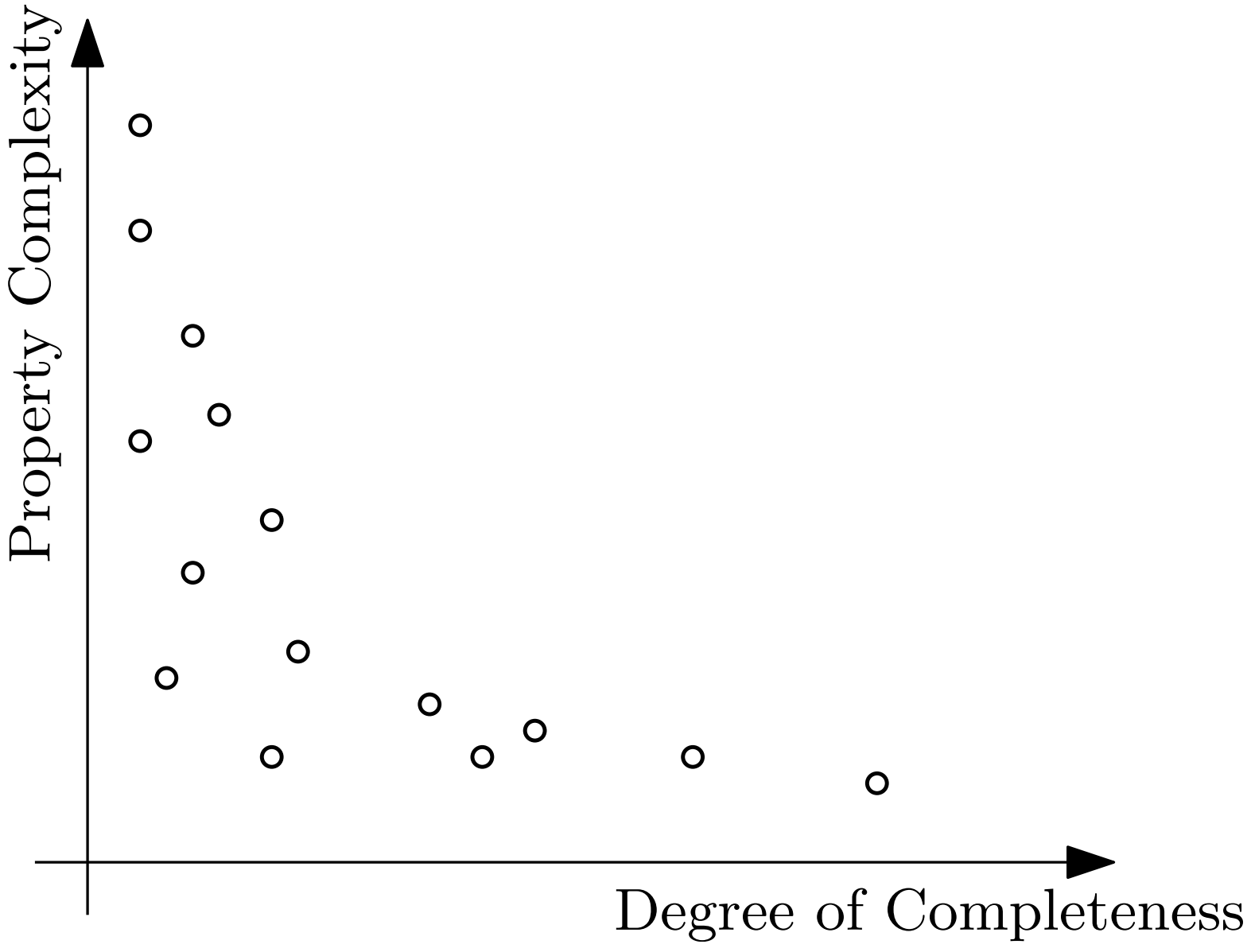
Program Logics



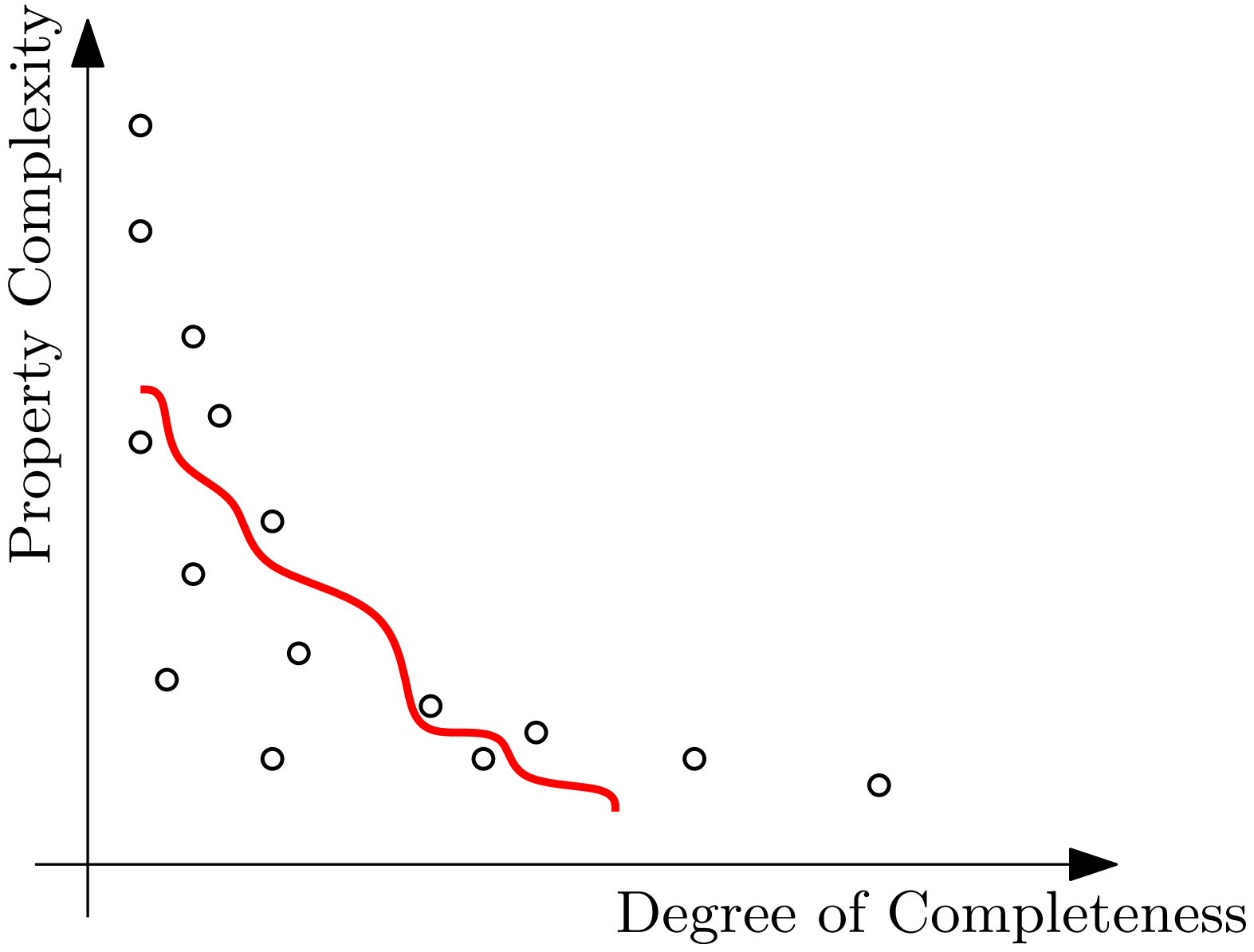
Program Logics



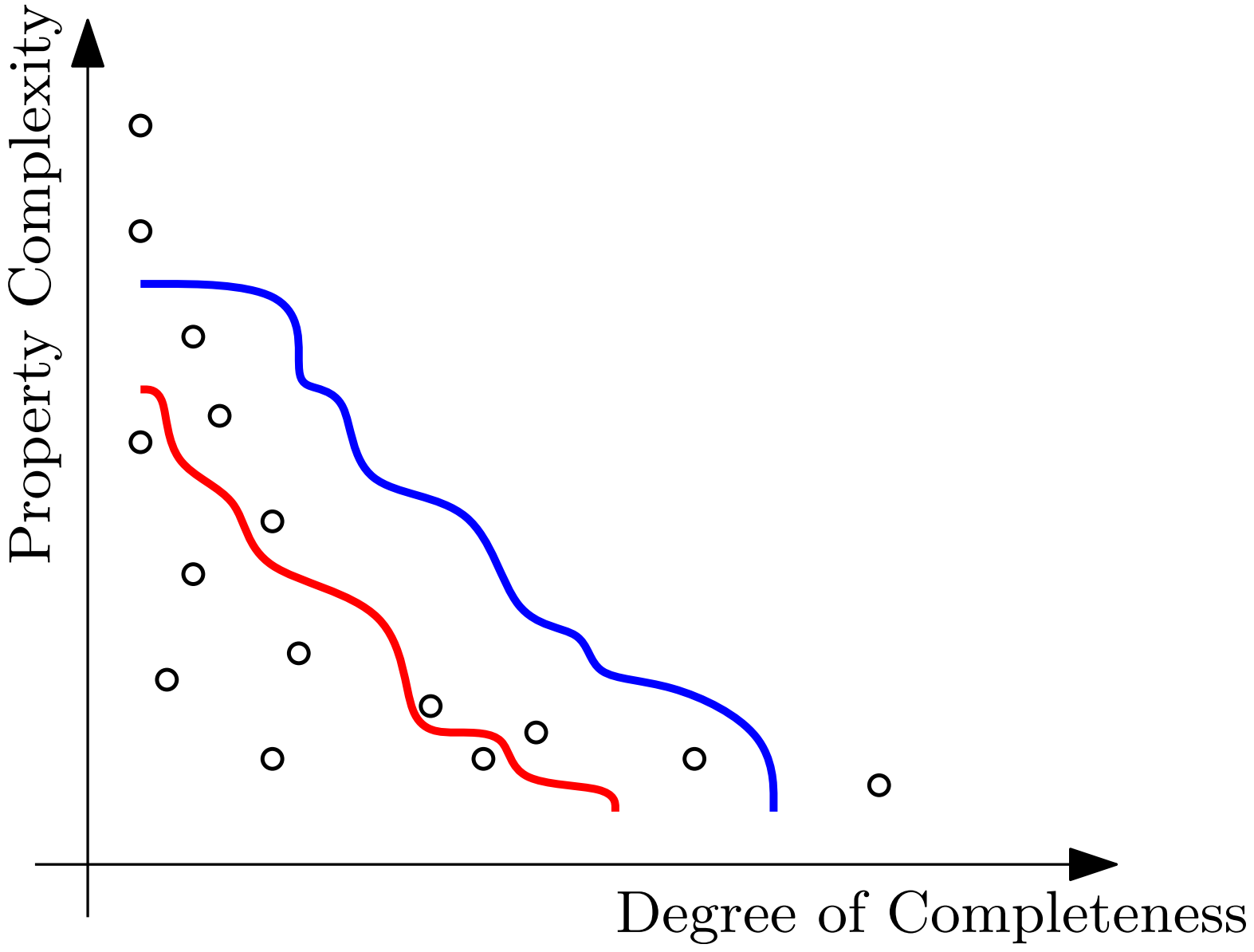
Type Systems



Type Systems



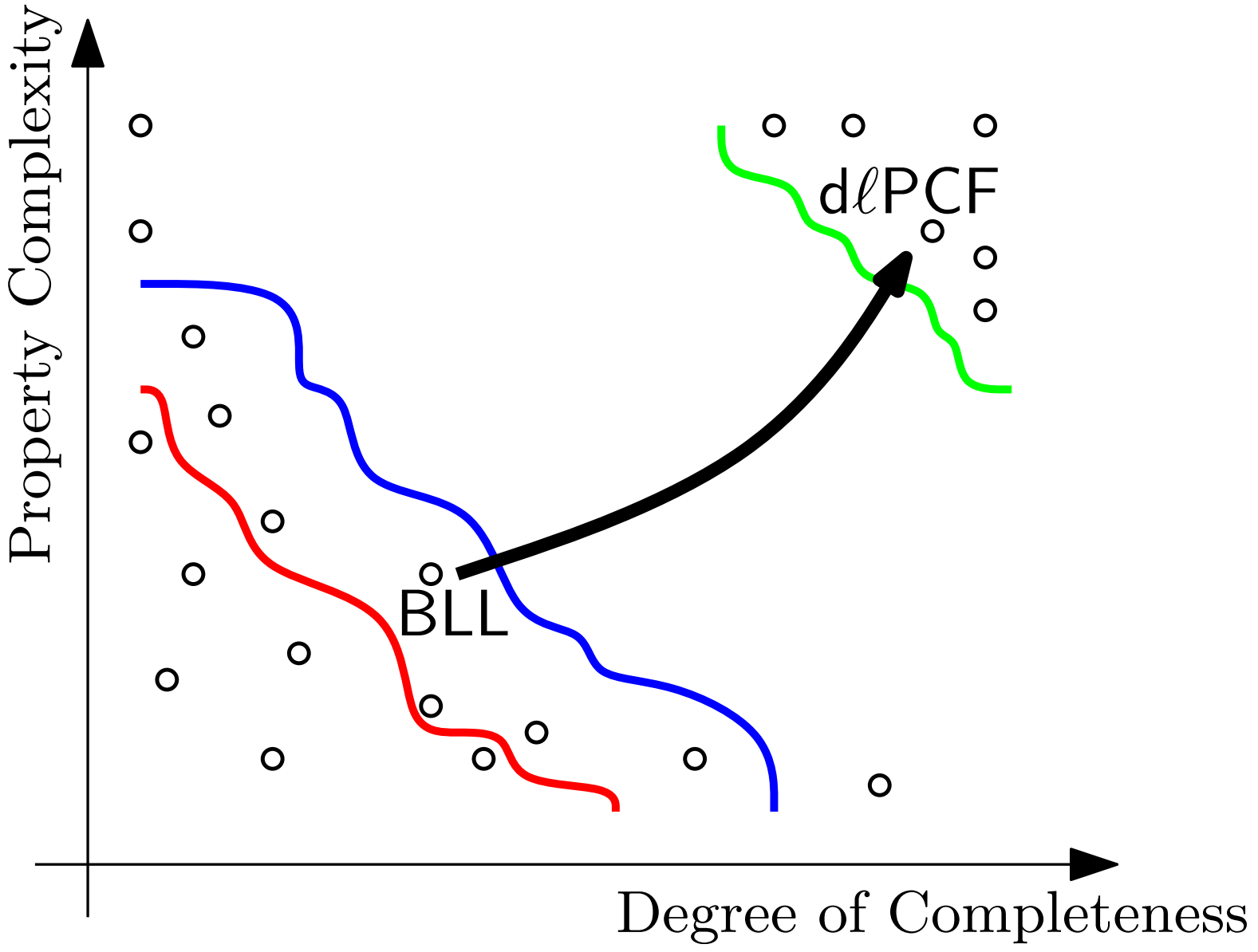
Type Systems



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



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?

- 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?

- 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?

- 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?

- 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.

Part III

dℓPCF

dℓPCF: a Bird's Eye View

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

$$\phi; \Phi \models^{\mathcal{E}} \mathrm{I} \leqslant \mathrm{J}$$

- Types and modal types are defined as follows:

$$\sigma, \tau ::= \mathtt{Nat}[\mathrm{I}, \mathrm{J}] \mid A \multimap \sigma \qquad \text{basic types}$$

$$A, B ::= [a < \mathrm{I}] \cdot \sigma \qquad \text{modal types}$$

dℓPCF: a Bird's Eye View

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

$$\phi; \mathrm{K}_1 \leqslant \mathrm{H}_1, \ldots, \mathrm{K}_n \leqslant \mathrm{H}_n \models^{\mathcal{E}} \mathrm{I} \leqslant \mathrm{J}$$

- Types and modal types are defined as follows:

$$\sigma, \tau ::= \mathtt{Nat}[\mathrm{I}, \mathrm{J}] \mid A \multimap \sigma \qquad \text{basic types}$$

$$A, B ::= [a < \mathrm{I}] \cdot \sigma \qquad \text{modal types}$$

The Meaning of Types

$$[a < \mathrm{I}] \cdot \sigma \multimap \tau$$

$$\Downarrow$$

$$(\sigma\{0/a\} \otimes \ldots \otimes \sigma\{\mathrm{I} - 1/a\}) \multimap \tau$$

The Meaning of Types

$$[a < \mathrm{I}] \cdot \sigma \multimap \tau$$

$$\Downarrow$$

$$(\sigma\{0/a\} \otimes \ldots \otimes \sigma\{\mathrm{I} - 1/a\}) \multimap \tau$$

dℓPCF: Subtyping

$$\frac{\phi; \Phi \models^{\mathcal{E}} \mathrm{K} \leqslant \mathrm{I} \qquad \phi; \Phi \models^{\mathcal{E}} \mathrm{J} \leqslant \mathrm{H}}{\phi; \Phi \vdash^{\mathcal{E}} \mathtt{Nat}[\mathrm{I}, \mathrm{J}] \sqsubseteq \mathtt{Nat}[\mathrm{K}, \mathrm{H}]}$$

$$\frac{\phi; \Phi \vdash^{\mathcal{E}} B \sqsubseteq A \qquad \phi; \Phi \vdash^{\mathcal{E}} \sigma \sqsubseteq \tau}{\phi; \Phi \vdash^{\mathcal{E}} A \multimap \sigma \sqsubseteq B \multimap \tau}$$

$$\frac{\phi, a; \Phi, a < \mathrm{J} \vdash^{\mathcal{E}} \sigma \sqsubseteq \tau \qquad \phi; \Phi \models^{\mathcal{E}} \mathrm{J} \leqslant \mathrm{I}}{\phi; \Phi \vdash^{\mathcal{E}} [a < \mathrm{I}] \cdot \sigma \sqsubseteq [a < \mathrm{J}] \cdot \tau}$$

dℓPCF: Subtyping

$$\frac{\phi; \Phi \models^{\mathcal{E}} \mathrm{K} \leqslant \mathrm{I} \qquad \phi; \Phi \models^{\mathcal{E}} \mathrm{J} \leqslant \mathrm{H}}{\phi; \Phi \vdash^{\mathcal{E}} \mathtt{Nat}[\mathrm{I}, \mathrm{J}] \sqsubseteq \mathtt{Nat}[\mathrm{K}, \mathrm{H}]}$$

$$\frac{\phi; \Phi \vdash^{\mathcal{E}} B \sqsubseteq A \qquad \phi; \Phi \vdash^{\mathcal{E}} \sigma \sqsubseteq \tau}{\phi; \Phi \vdash^{\mathcal{E}} A \multimap \sigma \sqsubseteq B \multimap \tau}$$

$$\frac{\phi, a; \Phi, a < \mathrm{J} \vdash^{\mathcal{E}} \sigma \sqsubseteq \tau \qquad \phi; \Phi \models^{\mathcal{E}} \mathrm{J} \leqslant \mathrm{I}}{\phi; \Phi \vdash^{\mathcal{E}} [a < \mathrm{I}] \cdot \sigma \sqsubseteq [a < \mathrm{J}] \cdot \tau}$$

dℓPCF: Some Rules

Constraints

$$\frac{\phi; \Phi \vdash^{\mathcal{E}} [a < \mathrm{I}] \cdot \sigma \sqsubseteq [a < \mathbf{1}] \cdot \tau}{\phi; \Phi; \Gamma, x : [a < \mathrm{I}] \cdot \sigma \vdash^{\mathcal{E}}_{\mathsf{0}} x : \tau\{\mathsf{0}/a\}}\ V$$

Weight

dℓPCF: Some Rules

$$\frac{\begin{array}{c}\phi;\Phi \vdash^{\mathcal{E}} \mathtt{Nat}[\mathrm{I}+1,\mathrm{J}+1] \sqsubseteq \mathtt{Nat}[\mathrm{K},\mathrm{H}] \\ \phi;\Phi;\Gamma \vdash^{\mathcal{E}}_{\mathrm{L}} t : \mathtt{Nat}[\mathrm{I},\mathrm{J}]\end{array}}{\phi;\Phi;\Gamma \vdash^{\mathcal{E}}_{\mathrm{L}} \mathtt{s}(t) : \mathtt{Nat}[\mathrm{K},\mathrm{H}]}\ S$$

dℓPCF: Some Rules

$$\frac{\phi; \Phi; \Gamma, x : [a < \mathrm{I}] \cdot \sigma \vdash^{\mathcal{E}}_{\mathrm{J}} t : \tau}{\phi; \Phi; \Gamma \vdash^{\mathcal{E}}_{\mathrm{J}} \lambda x.t : [a < \mathrm{I}] \cdot \sigma \multimap \tau}\ L$$

dℓPCF: Some Rules

$$\frac{\begin{array}{c}\phi;\Phi \vdash^{\mathcal{E}} \Sigma \sqsubseteq \Gamma \uplus \sum_{a<\mathrm{I}} \Delta \\ \phi;\Phi;\Gamma \vdash^{\mathcal{E}}_{\mathrm{J}} t : [a<\mathrm{I}] \cdot \sigma \multimap \tau \\ \phi,a;\Phi,a<\mathrm{I};\Delta \vdash^{\mathcal{E}}_{\mathrm{K}} u : \sigma\end{array}}{\phi;\Phi;\Sigma \vdash^{\mathcal{E}}_{\mathrm{J}+\sum_{a\leqslant \mathrm{I}} \mathrm{K}+\mathrm{I}} tu : \tau}\ A$$

dℓPCF: Some Rules

Sum of Modal Types

$$\frac{\begin{array}{c}\phi;\Phi \vdash^{\mathcal{E}} \Sigma \sqsubseteq \Gamma \uplus \sum_{a<\mathrm{I}} \Delta \\ \phi;\Phi;\Gamma \vdash^{\mathcal{E}}_{\mathrm{J}} t : [a<\mathrm{I}] \cdot \sigma \multimap \tau \\ \phi, a;\Phi, a<\mathrm{I};\Delta \vdash^{\mathcal{E}}_{\mathrm{K}} u : \sigma\end{array}}{\phi;\Phi;\Sigma \vdash^{\mathcal{E}}_{\mathrm{J}+\sum_{a\leqslant \mathrm{I}} \mathrm{K}+\mathrm{I}} tu : \tau}\ A$$

dℓPCF: Some Rules

Bounded Sum of Modal Types

$$\frac{\begin{array}{c}\phi;\Phi \vdash^{\mathcal{E}} \Sigma \sqsubseteq \Gamma \uplus \sum_{a<\mathrm{I}} \Delta \\ \phi;\Phi;\Gamma \vdash^{\mathcal{E}}_{\mathrm{J}} t : [a<\mathrm{I}] \cdot \sigma \multimap \tau \\ \phi, a;\Phi, a<\mathrm{I};\Delta \vdash^{\mathcal{E}}_{\mathrm{K}} u : \sigma\end{array}}{\phi;\Phi;\Sigma \vdash^{\mathcal{E}}_{\mathrm{J}+\sum_{a\leqslant \mathrm{I}} \mathrm{K}+\mathrm{I}} tu : \tau}\ A$$

dℓPCF: Intended Meaning

$$a; \varnothing; \varnothing \vdash_{\mathrm{I}} t : [b < \mathrm{J}] \cdot \mathtt{Nat}[a] \multimap \mathtt{Nat}[\mathrm{K}]$$

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 $\mathrm{K}\{n/a\}$
- Something more **intensional**:
  - The cost of evaluation of t on an input n is $(\mathrm{I} + \mathrm{J})\{n/a\}$.
- Two questions:
  - Is this **correct**?
  - How many programs can be captured this way?

dℓPCF: Intended Meaning

$$a; \varnothing; \varnothing \vdash_{\mathrm{I}} t : [b < \mathrm{J}] \cdot \mathtt{Nat}[a] \multimap \mathtt{Nat}[\mathrm{K}]$$

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 $\mathrm{K}\{n/a\}$
- Something more **intensional**:
  - The cost of evaluation of t on an input n is $(\mathrm{I} + \mathrm{J})\{n/a\}$.
- Two questions:
  - Is this **correct**?
  - How many programs can be captured this way?

dℓPCF: Intended Meaning

$$a; \varnothing; \varnothing \vdash_{\mathrm{I}} t : [b < \mathrm{J}] \cdot \mathtt{Nat}[a] \multimap \mathtt{Nat}[\mathrm{K}]$$

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 $\mathrm{K}\{n/a\}$
- Something more **intensional**:
  - The cost of evaluation of t on an input n is $(\mathrm{I} + \mathrm{J})\{n/a\}$.
- Two questions:
  - Is this **correct**?
  - How many programs can be captured this way?

dℓPCF: Intended Meaning

$$a; \varnothing; \varnothing \vdash_{\mathrm{I}} t : [b < \mathrm{J}] \cdot \mathtt{Nat}[a] \multimap \mathtt{Nat}[\mathrm{K}]$$

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 $\mathrm{K}\{n/a\}$
- Something more **intensional**:
  - The cost of evaluation of t on an input n is $(\mathrm{I} + \mathrm{J})\{n/a\}$.
- 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 $(t, \epsilon, \epsilon) \to^* D \to E$ *and let* D *have weight* I. *Then one of the following holds:*

1. E *has weight* J, $\phi; \Phi \models \mathrm{I} = \mathrm{J}$ *but* $|D| > |E|$;
2. E *has weight* J, $\phi; \Phi \models \mathrm{I} > \mathrm{J}$ *and* $|E| < |D| + |t|$;

Theorem

Let $\varnothing; \varnothing; \varnothing \vdash_{\mathrm{I}} t : \mathtt{Nat}[\mathrm{J}, \mathrm{K}]$ *and* $t \Downarrow^n \underline{\mathtt{m}}$. *Then* $n \leqslant |t| \cdot [\![\mathrm{I}]\!]^{\mathcal{E}}_{\rho}$

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

1. E *has weight* J, $\phi; \Phi \models \mathrm{I} = \mathrm{J}$ *but* $|D| > |E|$;
2. E *has weight* J, $\phi; \Phi \models \mathrm{I} > \mathrm{J}$ *and* $|E| < |D| + |t|$;

Theorem

Let $\varnothing; \varnothing; \varnothing \vdash_{\mathrm{I}} t : \mathtt{Nat}[\mathrm{J}, \mathrm{K}]$ *and* $t \Downarrow^n \underline{\mathtt{m}}$. *Then* $n \leqslant |t| \cdot [\![\mathrm{I}]\!]^{\mathcal{E}}_{\rho}$

Completeness for Programs

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

Lemma (Weighted Subject Expansion)

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

Theorem (Relative Completeness for Programs)

Let t be a PCF program such that $t \Downarrow^n \underline{\mathtt{m}}$. Then, there exist two index terms I and J such that $[\![\mathrm{I}]\!]^{\mathcal{U}} \leqslant n$ and $[\![\mathrm{J}]\!]^{\mathcal{U}} = m$ and such that the term t is typable in dℓPCF as $\varnothing; \varnothing; \varnothing \vdash^{\mathcal{U}}_{\mathrm{I}} t : \mathtt{Nat}[\mathrm{J}]$.

Completeness for Programs

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

Lemma (Weighted Subject Expansion)

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

Theorem (Relative Completeness for Programs)

Let t be a PCF program such that $t \Downarrow^n \underline{\mathtt{m}}$. Then, there exist two index terms I and J such that $[\![\mathrm{I}]\!]^{\mathcal{U}} \leqslant n$ and $[\![\mathrm{J}]\!]^{\mathcal{U}} = m$ and such that the term t is typable in dℓPCF as $\varnothing; \varnothing; \varnothing \vdash^{\mathcal{U}}_{\mathrm{I}} t : \mathtt{Nat}[\mathrm{J}]$.

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 $\vdash t : \mathtt{Nat} \to \mathtt{Nat}$. Moreover, suppose that there are two (total and computable) functions $f, g : \mathbb{N} \to \mathbb{N}$ such that $t\,\underline{\mathtt{n}} \Downarrow^{g(n)} \underline{\mathtt{f(n)}}$. Then there are terms $\mathrm{I}, \mathrm{J}, \mathrm{K}$ with $[\![\mathrm{I} + \mathrm{J}]\!] \leqslant g$ and $[\![\mathrm{K}]\!] = \overline{f}$, such that

$$a; \varnothing; \varnothing \vdash_{\mathrm{I}}^{\mathcal{U}} t : [b < \mathrm{J}] \cdot \mathtt{Nat}[a] \multimap \mathtt{Nat}[\mathrm{K}].$$

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.

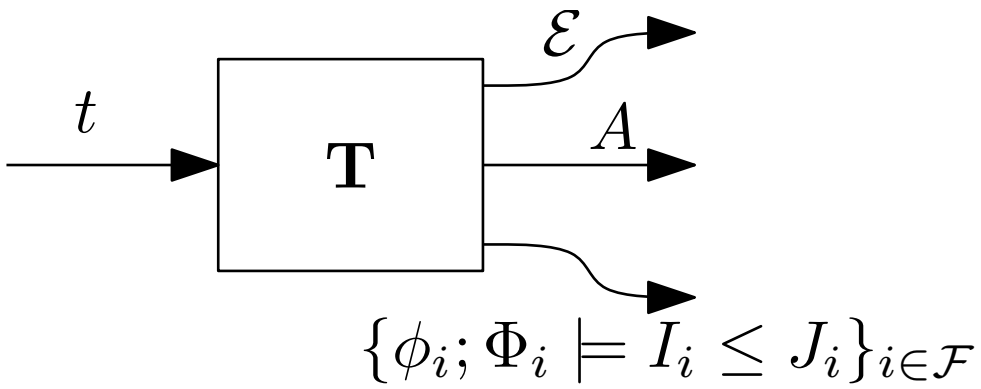
Theorem (Relative Completeness for Functions)

Suppose that t is a PCF *term such that $\vdash t : \mathtt{Nat} \to \mathtt{Nat}$. Moreover, suppose that there are two (total and computable) functions $f, g : \mathbb{N} \to \mathbb{N}$ such that $t\ \underline{\mathtt{n}} \Downarrow^{g(n)} \underline{\mathtt{f(n)}}$. Then there are terms $\mathrm{I}, \mathrm{J}, \mathrm{K}$ with $[\![\mathrm{I} + \mathrm{J}]\!] \leqslant g$ and $[\![\mathrm{K}]\!] = \overline{f}$, such that*

$$a; \varnothing; \varnothing \vdash^{\mathcal{U}}_{\mathrm{I}} t : [b < \mathrm{J}] \cdot \mathtt{Nat}[a] \multimap \mathtt{Nat}[\mathrm{K}].$$

Conclusions

- A relatively complete type system $\mathsf{d}\ell\mathsf{PCF}$.
- Type inference, type checking and derivation checking are undecidable, in general.
  - ... but can become manageable if $\mathcal{E}$ is simple enough.
  - Light Logics!
- Another result: relative decidability of type inference.



Thank you!

Questions?