Implicit Computational Complexity
and the quest for intensional completeness

Simone Martini

Pisa Summer Workshop on Proof Theory
June 13th, 2012
Outline

1. Proof of what?
2. Implicit Computational Complexity, ICC
3. Intermezzo: cost models for proof reduction
4. Intensional completeness
5. Intersection types
6. Program logics
7. Linear Dependent Types
1. Proof of what?
2. Implicit Computational Complexity, ICC
3. Intermezzo: cost models for proof reduction
4. Intensional completeness
5. Intersection types
6. Program logics
7. Linear Dependent Types
Most proof theory: combinatorial properties of proofs
Most such properties:
do not depend on being proofs of something
But only on their syntactical structure

E.g.: (weak/strong) normalization: Jervell’s yesterday talk!

“A wrong proof is not a proof. Incorrect reasoning still is reasoning.”
Most proof theory: combinatorial properties of proofs
Most such properties:
do not depend on being proofs of something
But only on their syntactical structure

E.g.: (weak/strong) normalization: Jervell’s yesterday talk!

“A wrong proof is not a proof.
Incorrect reasoning still is reasoning.”
The underlying realm

- Proofs are the tip of the iceberg
- They emerge from a realm of objects whose dynamics is relevant and complex
- Logic select among them some objects
- Untyped $\lambda$-terms; untyped proof-nets; ludics; ...
- LJ, F, F^ω, ... select increasing sets of normalizing terms

- To get all normalizing terms we need intersection types

- \( \Sigma_1^0 \)-complete set of terms

- “Logic”:
  - intensional: if \( M : A \) and \( M : B \), then \( M : A \land B \)
  - undecidable: \( M : A \)

- Decidable approximations...
λ-terms

- LJ, F, Fω, ... select increasing sets of normalizing terms

- To get all normalizing terms we need intersection types

- \( \Sigma^0_1 \)-complete set of terms

- “Logic”:
  - intensional: if \( M : A \) and \( M : B \), then \( M : A \land B \)
  - undecidable: \( M : A \)

- Decidable approximations...
\(\lambda\)-terms

- LJ, F, F^\omega, \ldots
  select increasing sets of normalizing terms

- To get all normalizing terms we need
  intersection types

- \(\Sigma^0_1\)-complete set of terms

- “Logic”:
  intensional: if \(M : A\) and \(M : B\), then \(M : A \land B\)
  undecidable: \(M : A\)

- Decidable approximations...
1 Proof of what?

2 Implicit Computational Complexity, ICC

3 Intermezzo: cost models for proof reduction

4 Intensional completeness

5 Intersection types

6 Program logics

7 Linear Dependent Types
Implicit Computational Complexity

- **Describe:**
  complexity phenomena, using *language restrictions*

- **Avoid:**
  external measure conditions, or explicit machine models.

- **Aim:**
  formal methods in software development; programming language design.

- **Use:**
  Logic
    - Model Theory (especially finite);
    - Recursion Theory;
    - **Proof Theory** (via Curry-Howard).
Bounded recursion on notation

- Bennett (1962) and Cobham (1965).
- A function \( f : \mathbb{N}^{n+1} \to \mathbb{N} \) is defined by bounded recursion on notation from \( g_0, g_1 : \mathbb{N}^n \to \mathbb{N}, h_0, h_1 : \mathbb{N}^{n+2} \to \mathbb{N} \) and \( k : \mathbb{N}^{n+1} \to \mathbb{N} \) if

\[
\begin{align*}
  f(0, \bar{y}) &= g_0(\bar{y}) \\
  f(1, \bar{y}) &= g_1(\bar{y}) \\
  f(s_0(x), \bar{y}) &= h_0(x, \bar{y}, f(x, \bar{y})) \\
  f(s_1(x), \bar{y}) &= h_1(x, \bar{y}, f(x, \bar{y}))
\end{align*}
\]

provided \( f(x, \bar{y}) \leq k(x, \bar{y}) \).
- In the initial functions: \( x \# y = 2^{\max(|x|, |y|)} \)
- note: \( |x|^k = |x| \# \cdots \# |x| \).
Safe recursion on notation

- Bellantoni & Cook (1992)
- The function $f$ is defined by safe recursion on notation from $g_0, g_1, h_0, h_1$ if

\[
\begin{align*}
  f(0, \bar{x}; \bar{y}) &= g_0(\bar{x}; \bar{y}) \\
  f(1, \bar{x}; \bar{y}) &= g_1(\bar{x}; \bar{y}) \\
  f(s_0(x), \bar{x}; \bar{y}) &= h_0(x, \bar{x}; \bar{y}, f(x, \bar{x}; \bar{y})) \\
  f(s_1(x), \bar{x}; \bar{y}) &= h_1(x, \bar{x}; \bar{y}, f(x, \bar{x}; \bar{y}))
\end{align*}
\]
Light Logics

For some proof systems (eg, fragments of Linear Logic, like LAL):

- Let $\Pi$ be a proof in the system.
- Then $\Pi$ can be reduced to normal form in time bounded by

$$O((d + 1) \cdot |\Pi|^{2^d+1}),$$

where $d$ is a certain parameter of $\Pi$.
- $d$ is shared by many different proofs

When $d$ is fixed, this is a polynomial in $|\Pi|$.

- Eg: for some $A$, all normal proofs $a : A$ have the same $d$;
  fixed $F : A \rightarrow B$, all proofs $Fa$ have the same $d$,
  hence normalize in polytime in $|a|$.
For some proof systems (e.g., fragments of Linear Logic, like LAL):

- Let $\Pi$ be a proof in the system.
- Then $\Pi$ can be reduced to normal form in time bounded by
  \[ O((d + 1) \cdot |\Pi|^{2^{d+1}}), \]
  where $d$ is a certain parameter of $\Pi$.
- $d$ is shared by many different proofs.

When $d$ is fixed, this is a polynomial in $|\Pi|$.

E.g.: for some $A$, all normal proofs $a : A$ have the same $d$; fixed $F : A \multimap B$, all proofs $Fa$ have the same $d$, hence normalize in polytime in $|a|$.
Proof theory vs Recursion theory

- Safe recursion: extensional definition of a class
- Indeed: rewriting is exponentially long
- Unless: call by value is used (Beckman & Weierman, 1996)

- Light logics have a built-in computational engine: proof reduction

- What kind of reduction
cut-elimination, $\beta$-reduction, proof-net normalization
is a delicate matter

- An intensional analysis...
Characterizing Complexity Classes

$L$ $C$
Characterizing Complexity Classes

Diagram showing a relationship between sets \( L \) and \( C \).
Characterizing Complexity Classes

\[ \mathcal{L}, \mathcal{C} \]
Characterizing Complexity Classes

Soundness: $\mathcal{S} \subseteq \mathcal{P}$
Completeness: $\mathcal{S} \supseteq \mathcal{P}$
Completeness, extensional

\[ S \supseteq \mathcal{P} : \]

For every function \( f \) which can be computed within the bounds of \( \mathcal{P} \), there is \( P \in S \) such that \( [P] = f \).
Soundness

\([S] \subseteq \mathcal{P} :\)

- **Semantically**
  - For every \(P \in S\), *some* algorithm computing \([P]\) exists which works within the prescribed resource bounds.
  - \(P \in \mathcal{L}\) does *not* necessarily have a nice computational behavior.
  - Examples: BC; LAL vs DLAL;
    soundness by realizability [DalLago&Hofmann05].

- **Operationally**
  - \(\mathcal{L}\) has an effective operational semantics.
  - Fix \(\mathcal{L}_\mathcal{P} \subseteq \mathcal{L}\): set of those programs reducing within the bounds of \(C\).
  - \([S] \subseteq \mathcal{P}\) can be shown by proving \(S \subseteq \mathcal{L}_\mathcal{P}\).
LAL vs DLAL

- **LAL** (Light Affine Logic) extensionally characterizes polytime. Any LAL proof-net may be reduced in polytime.

- Essential: some portions (scopes, boxes) are enclosed in safety boxes.

- We may interpret LAL proofs (skeletons) also as pure $\lambda$-terms: $\Gamma \vdash M : A$

- Some of these $\lambda$-terms reduce in exp-time!

- Boxes are needed during reduction to ensure polytime

- Design a restricted system DLAL (Baillot & Terui)
Outline

1. Proof of what?
2. Implicit Computational Complexity, ICC
3. Intermezzo: cost models for proof reduction
4. Intensional completeness
5. Intersection types
6. Program logics
7. Linear Dependent Types
The question

What is a good *cost model* for proof reduction, taking into account *(only)* the intrinsic description of the language, and not *(also)* its implementation on a conventional machine?

where

*In proof reduction, the elementary computation step (*β*-reduction, cut elimination, etc.) is *not* a constant-time operation.*
What we do not want

*Deus ex machina*

The cost accounted for an entire computation is the cost needed to simulate it on a *Turing machine*

with no general (or uniform) relation between the “intrinsic” (but non constant-time) elementary steps and the cumulative number of steps of the TM.
A good cost model. . .

. . . is polynomially related (or invariant) to the cost as computed on a Turing machine

There is a polynomial $p$ such that the cost of computing (the normal form of) $M$ under the cost model $c$ is

$$\text{Cost}_c(M) \leq p(\text{Cost}_{TM}(M))$$

For $f$ computed by a Turing machine $M$ in time $g$, there is a program $N_M$ computing $f$ in $\text{Cost}_c(O(g(n)))$. 
Weak call-by-value $\lambda$-calculus

- **Terms**
  \[ M ::= x \mid \lambda x. M \mid MM \]

- **Values**
  \[ V ::= x \mid \lambda x. M \]

- Weak call-by-value reduction

\[
\begin{align*}
(\lambda x. M)V & \rightarrow_v M\{V/x\} \\
ML & \rightarrow_v NL \\
LM & \rightarrow_v LN
\end{align*}
\]
Explicit representation: the difference cost model

- If terms are represented explicitly as strings
- In particular, we want to print the result as a string

Difference cost model:
for each $\beta$-step $M \rightarrow_v N$, count $\max\{1, |N| - |M|\}$

The difference cost model is polynomially invariant for weak reduction (by value or by name).

(Dal Lago and M., CiE 2006)
Implicit representation: the unitary cost model

- If we allow shared (graph) representation for terms
- In particular, the result could be a shared graph

- Unitary cost model:
  for each $\beta$-step $M \rightarrow_v N$, count 1

_The unitary cost model is polynomially invariant for weak reduction (by value or by name)._
Implicit representation 2: the unitary cost model

Using other compact representations: explicit substitutions

The unitary cost model is polynomially invariant for head reduction

(Accattoli and Dal Lago, RTA 2012)
ICC: Intensional Expressivity
ICC: Intensional Expressivity
ICC: Intensional Expressivity
ICC: Intensional Expressivity
ICC: Intensional Expressivity
The set of programs (TMs, pure \( \lambda \)-terms, TRS, . . .) reducing in polytime in the size of their argument is \( \Sigma^0_2 \)-complete. . .

(Hajék 1979)
Some Examples

- **Simple 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.
Outline

1. Proof of what?
2. Implicit Computational Complexity, ICC
3. Intermezzo: cost models for proof reduction
4. Intensional completeness
5. Intersection types
6. Program logics
7. Linear Dependent Types
Intersection types

- $T ::= \phi \mid T \rightarrow T \mid T \land T$

- \[ \frac{M : A}{M : A \land B} \quad \frac{M : A \land B}{M : A} \quad \frac{M : A \land B}{M : B} \]

- Several equations among types, especially:

  \[ A \land A \equiv A \]
Non idempotent intersection types

- Drop: \( A \land A \equiv A \)
- \( M : A \land B \)
  - From: “\( A \) may be used as data of type \( A \) or of type \( B \)”.
  - To: “\( A \) may be used once as data of type \( A \), and once as data of type \( B \)”.

- \( M \) is SN iff \( M \) is typeable

- The length of longest reduction of a normalizable term \( M \) can be read off its typing derivation.

[Bernadet & Lengrand 2011; also de Carvalho 2009]
Non idempotent intersection types

- **Drop:** $A \land A \equiv A$
- **$M : A \land B$**
  - From: “$A$ may be used as data of type $A$ or of type $B$”.
  - To: “$A$ may be used once as data of type $A$, and once as data of type $B$”.

- $M$ is SN iff $M$ is typeable

- The **length** of longest reduction of a normalizable term $M$ can be read off its typing derivation.
  
  [Bernadet & Lengrand 2011; also de Carvalho 2009]
Non uniform definability

Which functions are definable in intersection types?

Uniform vs Non uniform definability

- Uniform:
  intersection types define the same functions than simple types
  [conj.: Leivant 1990; proof: Bucciarelli et al. 2003]

- Non uniform:
  all total functions (a $\Pi_2$-complete set).
  Different derivations for different inputs.
  Highly non-uniform type derivations for the same function

How to design decidable approximations?
Non uniform definability

- Which functions are definable in intersection types?

- **Uniform vs Non uniform** definability
  - **Uniform:**
    intersection types define the same functions than simple types
    [conj.: Leivant 1990; proof: Bucciarelli *et al.* 2003]
  - **Non uniform:**
    all total functions (a $\Pi_2$-complete set).
    - Different derivations for different inputs.
    - Highly non-uniform type derivations for the **same** function

- How to design **decidable approximations**?
Outline

1. Proof of what?
2. Implicit Computational Complexity, ICC
3. Intermezzo: cost models for proof reduction
4. Intensional completeness
5. Intersection types
6. Program logics
7. Linear Dependent Types
Program Logics

- Judgments:

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

precondition \hspace{2cm} postcondition

program

- Some rules:

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

\[
\begin{array}{c}
\{P\} \quad C \quad \{Q\} \\
\{Q\} \quad D \quad \{R\} \\
\{P\} \quad C;D \quad \{R\}
\end{array}
\]

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

- Judgments:

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

precondition \rightarrow program \rightarrow postcondition

- Some rules:

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

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

\[
R \Rightarrow P \quad \{ P \} C \{ Q \} \quad Q \Rightarrow S \quad \{ R \} C \{ S \}
\]
Program Logics: Relative Completeness

- **Sound:**
  - If true formulas of PA are used as side-conditions.

- **Relatively complete:**
  - All true assertions derived, if all true PA formulas used as side-conditions.

- Throw in a concrete sound formal system $\mathcal{F}$ for PA:
  - Sound.
  - Incomplete, by Gödel incompleteness.
  - $\mathcal{F}$ is the sole responsible for incompleteness.

- A variety of FH logics...
  - Including some for higher-order programs [Honda2000]
  - ...and some in which the complexity of programs is taken into account.
Outline

1 Proof of what?
2 Implicit Computational Complexity, ICC
3 Intermezzo: cost models for proof reduction
4 Intensional completeness
5 Intersection types
6 Program logics
7 Linear Dependent Types
Linear dependent types

- **Linearity:**
  - Control the times subterms are copied during evaluation

- **Dependency:**
  - Distinct copies of a term are typed with distinct types.
  - As in intersection types, but in a uniform way.

- **dℓPCF**, inspired by BLL

  Dal Lago & Gaboardi, LICS 2011
Bounded Linear Logic

- Extensionally complete for polytime functions [GSS1992].
- Types:
  \[ A ::= \alpha(p_1, \ldots, p_n) \mid A \otimes A \mid A \rightarrow A \mid \forall \alpha.A \mid !_{x<p} A \]
- In BLL there are many “polytime proofs” [DalLagoHofmann2010].
M : [a < I] · A → B

M uses its argument I times, each time with type A{n/a}, for 0 ≤ n < I.
\[ a; \emptyset; \emptyset \vdash_I t : [b < J] \cdot \text{Nat}[a] \rightarrow \text{Nat}[K] \]

- \( t \) computes a function from \( \text{Nat} \) to \( \text{Nat} \).

- **Extensional:**
  - On input a number \( n \), \( t \) returns a number \( K\{n/a}\)

- **Intensional:**
  - The cost of evaluation of \( t \) on an input \( n \) is \( (I + J)\{n/a}\).
\[ a; \emptyset; \emptyset \vdash I \frac{b}{J} \cdot \text{Nat}[a] \rightarrow \text{Nat}[K] \]

- \( t \) computes a function from \( \text{Nat} \) to \( \text{Nat} \).

- **Extensional:**
  - On input a number \( n \), \( t \) returns a number \( K\{n/a\} \)

- **Intensional:**
  - The cost of evaluation of \( t \) on an input \( n \) is \( (I + J)\{n/a\} \).
\[ \begin{align*}
\phi; \Phi & \vdash^\mathcal{E} K \leq I \\
\phi; \Phi & \vdash^\mathcal{E} J \leq H \\
\phi; \Phi & \vdash^\mathcal{E} \text{Nat}[I, J] \sqsubseteq \text{Nat}[K, H]
\end{align*} \]

\[ \begin{align*}
\phi; \Phi & \vdash^\mathcal{E} G \sqsubseteq F \\
\phi; \Phi & \vdash^\mathcal{E} A \sqsubseteq B \\
\phi; \Phi & \vdash^\mathcal{E} F \rightarrow A \sqsubseteq G \rightarrow B
\end{align*} \]

\[ \begin{align*}
\phi, a; \Phi, a < J & \vdash^\mathcal{E} A \sqsubseteq B \\
\phi; \Phi & \vdash^\mathcal{E} J \leq I \\
\phi; \Phi & \vdash^\mathcal{E} [a < I] \cdot A \sqsubseteq [a < J] \cdot B
\end{align*} \]
A variation on Krivine’s machine as abstract evaluator:
\[ t \Downarrow^n m. \]

**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.

**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 $[I]^{\mathcal{E}} \leq n$ and $[J]^U = m$ and such that the term $t$ is typable in $d\ell\text{PCF}$ as $\emptyset; \emptyset; \emptyset \vdash^\mathcal{E} t : \text{Nat}[J]$. 
Completeness for Functions

- It strongly relies on the universality of $\mathcal{E}$.
- Let $\{\pi_n\}_{n \in \mathbb{N}}$ be 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

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

- It strongly relies on the universality of $\mathcal{E}$.
- Let $\{\pi_n\}_{n \in \mathbb{N}}$ be 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 $t \nrightarrow g(n) f(n)$. Then there are terms $I, J, K$ with $[I + J] \leq g$ and $[K] = \bar{f}$, such that

$$a; \emptyset; \emptyset \vdash^U_I t : [b < J] \cdot \text{Nat}[a] \twoheadrightarrow \text{Nat}[K].$$
Decidable approximations?

\[ P \in \mathcal{L} \]

\[
\text{dℓPCF}
\]

\[
\begin{cases}
\text{Yes, } P \in \mathcal{LP} + \text{ bounds} \\
\text{Don’t know}
\end{cases}
\]
A better choice

\[ d\ell \text{PCF} \quad \xrightarrow{P \in \mathcal{L}} \quad \xrightarrow{B \text{Bound}} \quad \xrightarrow{P_B \in \mathcal{J}} \quad \xrightarrow{\text{ITP}} \quad \xrightarrow{\text{SMT}} \quad \xrightarrow{\text{AI}} \quad \ldots \]
A better choice

$P \downarrow_B \text{ iff } \models P_B$

$P \in \mathcal{L}$

Bound $B$

$P_B \in \mathcal{I}$

$\downarrow$PCF

$\models$

Bound

$\models$

$I\!T\!P$

$S\!M\!T$

$A\!I$

$\ldots$
Conclusions

- We started from *implicit* complexity

- We presented a relative (intensionally) complete system

- Much more... *explicit* than we expected at the start!
Conclusions

- We started from \textit{implicit} complexity
- We presented a relative (intensionally) complete system
- Much more...\textit{explicit} than we expected at the start!
“Applied proof-theory” *in partibus infidelium*
dℓPCF: Some Rules

Constraints

\[
\phi; \Phi \vdash^\varepsilon [a < I] \cdot A \subseteq [a < 1] \cdot B
\]

\[
\phi; \Phi; \Gamma, x : [a < I] \cdot A \vdash^\varepsilon x : B\{/a\}
\]

Weight
$\phi; \Phi \vdash^E \text{Nat}[I + 1, J + 1] \sqsubseteq \text{Nat}[K, H]$

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

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

$S$
dℓPCF: Some Rules

\[
\phi; \Phi; \Gamma, x : [a < I] \cdot A \vdash^\varepsilon_j t : B
\]

\[
\phi; \Phi; \Gamma \vdash^\varepsilon \lambda x. t : [a < I] \cdot A \to B
\]

\[ L \]
dℓPCF: Some Rules

\[
\begin{align*}
\phi; \Phi \vdash \varepsilon \sum \subseteq \Gamma \uplus \sum_{a < I} \Delta \\
\phi; \Phi; \Gamma \vdash \varepsilon \text{ } t : [a < I] \cdot A \rightarrow B \\
\phi, a; \Phi, a < I; \Delta \vdash \varepsilon \text{ } u : A \\
\phi; \Phi; \sum \vdash \varepsilon \text{ } J + \sum_{a \leq I} K + I \text{ } tu : B \\
\end{align*}
\]
dℓPCF: Some Rules

Sum of Modal Types

\[ \phi; \Phi \vdash^\epsilon \Sigma \subseteq \Gamma \cup \sum_{a < I} \Delta \]

\[ \phi; \Phi; \Gamma \vdash^\epsilon t : [a < I] \cdot A \rightarrow B \]

\[ \phi, a; \Phi, a < I; \Delta \vdash^\epsilon_K u : A \]

\[ \phi; \Phi; \sum \vdash^\epsilon_{J+\sum_{a \leq I} K+I} tu : B \]
dℓPCF: Some Rules

Bounded Sum of Modal Types

\[ \phi; \Phi \vdash \varepsilon \sum \subseteq \Gamma \cup \sum_{a < I} \Delta \]
\[ \phi; \Phi; \Gamma \vdash \varepsilon \lambda t : [a < I] \cdot A \rightarrow B \]
\[ \phi, a; \Phi, a < I; \Delta \vdash \varepsilon K u : A \]
\[ \phi; \Phi; \sum \vdash \varepsilon J + \sum_{a \leq I} K + I tu : B \]
Theorem (Soundness)

If $\mathcal{T}I(P, \Phi) = (\sigma, I, \epsilon, \mathcal{I})$ and $\mathcal{F} \supseteq \epsilon$ is such that $\mathcal{F} \models \mathcal{I}$, then $\Phi \vdash_\mathcal{I} P : \sigma$.

Theorem (Completeness)

If $\Phi \vdash_\mathcal{I} P : \sigma$ and $\mathcal{T}I(P, \Phi) = (\tau, J, \mathcal{F}, \mathcal{I})$, then there is $\mathcal{G} \supseteq \mathcal{F}$ such that $\Phi \vdash_\mathcal{J} P : \tau$, $\mathcal{G} \cup \mathcal{F} \models I \geq J$ and $\mathcal{G} \cup \mathcal{F} \models \tau \sqsubseteq \sigma$. 