Implicit Computational Complexity

Simone Martini

Dipartimento di Scienze dell'Informazione
Università di Bologna
Italy

Bertinoro International Spring School
for Graduate Studies in Computer Science,
6–17 March, 2006

Outline: second part

Proof Theory

- Intuitionistic Logic and the Curry Howard Isomorphism

Logic and Programming Languages

Challenges

Second Part: Proof theory techniques

We shift from function classes to logical systems
We investigate computational "built-in" mechanisms
And learn how to cut them down to interesting complexity classes

To say the truth...
Already our approach to Gödel's T is not in the function algebra style.
We defined T as a formal system where there is a built-in computational mechanism (machine model): λ-calculus' beta reduction.
Next step will be to get rid of the base type of natural numbers and use "bare" logical systems.

Second Order Intuitionistic Logic, Sequent calculus

$$A \vdash A \ (Ax)$$

$$\frac{\Gamma \vdash A \quad A, \Delta \vdash B}{\Gamma, \Delta \vdash B} \ (Cut)$$

$$\frac{\Gamma \vdash C}{\Gamma, A \vdash C} \ (Weak.)$$

$$\frac{\Gamma, A, A \vdash B}{\Gamma, A \vdash B} \ (Contr.)$$

$$\frac{\Gamma \vdash A \quad B, \Delta \vdash C}{\Gamma, A \rightarrow B, \Delta \vdash C} \ (\rightarrow, l)$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \rightarrow B} \ (\rightarrow, r)$$

$$\frac{\Gamma, A, B \vdash C}{\Gamma, A \wedge B \vdash C} \ (\wedge, l)$$

$$\frac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma, \Delta \vdash A \wedge B} \ (\wedge, r)$$

$$\frac{\Gamma, T[S/t] \vdash C}{\Gamma, \forall t.T \vdash C} \ (\forall, l)$$

$$\frac{\Gamma \vdash C}{\Gamma \vdash \forall t.C} \ t \notin FV(\Gamma) \ (\forall, r)$$

The Curry-Howard correspondence: Annotated proofs

$$x : A \vdash x : A \ (Ax)$$

$$\frac{\Gamma \vdash M : A \quad x : A, \Delta \vdash N : B}{\Gamma, \Delta \vdash N[M/x] : B} \ (Cut)$$

$$\frac{\Gamma \vdash M : C}{\Gamma, x : A \vdash M : C} \ (Weak.)$$

$$\frac{\Gamma, y : A, z : A \vdash M : B}{\Gamma, x : A \vdash M[z/x, y/z] : B} \ (Contr.)$$

$$\frac{\Gamma \vdash M : A \quad x : B, \Delta \vdash N : C}{\Gamma, f : A \to B, \Delta \vdash N[fM/x] : C} \ (\to, l)$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \to B} \ (\to, r)$$

$$\frac{\Gamma, x : A, y : B \vdash M : C}{\Gamma, z : A \wedge B \vdash M[fz/x, sz/y] : C} \ (\wedge, l)$$

$$\frac{\Gamma \vdash M : A \quad \Delta \vdash N : B}{\Gamma, \Delta \vdash \langle M, N \rangle : A \wedge B} \ (\wedge, r)$$

$$\frac{\Gamma, x : T[S/t] \vdash M : C}{\Gamma, x : \forall t.T \vdash M : C} \ (\forall, l)$$

$$\frac{\Gamma \vdash M : C}{\Gamma \vdash M : \forall t.C} \ t \notin FV(\Gamma) \ (\forall, r)$$

The Curry-Howard correspondence: Computing with proofs

- Notion of normalization on proofs: cut elimination.
- We may annotate proofs with λ-terms.
- Normalization of proofs is β-reduction on λ-terms
- Expressiveness: Code natural numbers as a certain type $T_{\mathbb{N}}$; then study the functions definable by terms with type $T_{\mathbb{N}} \rightarrow T_{\mathbb{N}}$
- Complexity: study the cost of normalizing a term

Comparison with the "function algebra" setting

- Function algebras
  - Primitive notion: data types (binary strings) and the operations on them;
  - Control added as a form of rewriting
- Curry-Howard correspondence
  - Primitive notion: logical proofs and their normalization;
  - Datatypes added as specific formulas

Types and data in Second Order Intuitionistic Logic

- The annotated system is called System F
- Identity: $\lambda x^t.x : \forall t.t \to t$;
- Natural numbers: $\mathbb{N} = \forall t.(t \to t) \to (t \to t)$;
- The number 3: $\underline{3} = \lambda f^{t\to t}.\lambda x^t.f(f(fx)) : \mathbb{N}$
  These are the Church numerals. In general:
  $\underline{n} = \lambda f^{t\to t}.\lambda x^t.f^n x : \mathbb{N}$
- Binary words: $\mathbb{B} = \forall t.(t \to t) \to (t \to t) \to (t \to t)$;
- The binary word 01 (that is: $s_0 s_1 \epsilon$): $\lambda s_0^{t\to t}.\lambda s_1^{t\to t}.\lambda e^t.s_0(s_1 e)$;
- In general: any "inductive" free algebra can be expressed in this way (Berarducci & Böhm)

Computing with free algebras

- Elements of the free algebras behave like iterators over arbitrary data
- Examples in $\mathbb{N}$:
  - Let T be any type and let $F : T \to T$.
  - For any $a : T$ we have $\underline{n}\ F\ a \to F(F \cdots (Fa) \cdots)$, with n occurrences of F.
  - $iter_T = \lambda n.\lambda f.\lambda x.n\ f\ x : \mathbb{N} \to (T \to T) \to T \to T$.
  - A doubling function:
    $double = \lambda n.\underline{2}n : \mathbb{N} \to \mathbb{N}$;
  - An exponential function:
    $exp = \lambda n.iter_{\mathbb{N}}\ n\ double\ \underline{1} : \mathbb{N} \to \mathbb{N}$

Expressivity of System F

- Any term of System F is strongly normalizing (Girard, 1972);
- Very strong consistency result for second order arithmetic;
- An (extensional) function f from naturals to naturals is coded with a term $M_f : \mathbb{N} \rightarrow \mathbb{N}$ iff f is provably total in second order arithmetic.
- A huge class!
- Normalizing a term in System F requires hyperexponential time.

Harnessing the power of System F, I

- Restrict the language of types and/or the rules to compute with them.
- Ban the second order (i.e., polymorphic) types.
  The simply typed lambda-calculus
- With simple types, the class of representable functions is strongly influenced by the underlying coding scheme:
  - If we fix normal forms for $\mathbb{N}_0 = (\alpha \to \alpha) \to (\alpha \to \alpha)$ to be the only legal encoding of numerals, then the class of representable functions is very small (the extended polynomials of Schwichtenberg 1976)
  - We may relax this constraint, allowing for instances of $\mathbb{N}_0$
  - In general, even inside the simply-typed λ-calculus, normalization is costly: it is not even Kalmar elementary in the size of the term being normalized (Statman 1979).

Harnessing the power of System F, II

- A better approach is to change the underlining logical machinery
- In particular: limit the arbitrary duplication in a computation (proof)
- That is: control the contraction rule.
- The drastic removal of contraction and weakening gives as (multiplicative) Linear Logic (LL)
- LL has a fast (polytime) normalization procedure
- It has, however, too little expressive power.
- Hence, reintroduce controlled duplication in the form of modal annotations on formulas to be contracted.

Intuitionistic Multiplicative Linear Logic: IMLL

$$A \vdash A \; (Ax) \qquad \frac{\Gamma \vdash A \quad A, \Delta \vdash B}{\Gamma, \Delta \vdash B} \; (Cut)$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B} \; (\multimap, r) \qquad \frac{\Gamma \vdash A \quad B, \Delta \vdash C}{\Gamma, A \multimap B, \Delta \vdash C} \; (\multimap, l)$$

$$\frac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C} \; (\otimes_i, l) \qquad \frac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B} \; (\otimes, r)$$

$$\frac{\Gamma, T[S/t] \vdash C}{\Gamma, \forall t.T \vdash C} \; (\forall, l) \qquad \frac{\Gamma \vdash C}{\Gamma \vdash \forall t.C} \; t \notin FV(\Gamma) \; (\forall, r)$$

Proof-nets for Multiplicative Linear Logic

- Proof-nets are a graph notation for (sequent) proofs.
- Normalization is a simple local procedure of graph-rewriting, at least in the multiplicative case.
- In the multiplicative case the normalization is polynomial (actually linear in the size of the graph).
- But multiplicative logic is not expressive enough...
-
- Details on proof nets at recitation?

Adding Exponentials: I(ME)LL

$$A \vdash A \ (Ax) \qquad \frac{\Gamma \vdash A \quad A, \Delta \vdash B}{\Gamma, \Delta \vdash B} \ (Cut)$$

$$\frac{\Gamma \vdash C}{\Gamma, !A \vdash C} \ (Weak.) \qquad \frac{\Gamma, !A, !A \vdash B}{\Gamma, !A \vdash B} \ (Contr.)$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B} \ (\multimap, r) \qquad \frac{\Gamma \vdash A \quad B, \Delta \vdash C}{\Gamma, A \multimap B, \Delta \vdash C} \ (\multimap, l)$$

$$\frac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C} \ (\otimes_i, l) \qquad \frac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B} \ (\otimes, r)$$

$$\frac{\Gamma, A \vdash B}{\Gamma, !A \vdash B} \ (!, l) \qquad \frac{!A_1, \ldots, !A_n \vdash B}{!A_1, \ldots, !A_n \vdash !B} \ (!, r)$$

$$\frac{\Gamma, T[S/t] \vdash C}{\Gamma, \forall t.T \vdash C} \ (\forall, l) \qquad \frac{\Gamma \vdash C}{\Gamma \vdash \forall t.C} \ t \notin FV(\Gamma) \ (\forall, r)$$

Proof nets for Multiplicative Exponential Linear Logic

Something at recitation?

A variant

$$A \vdash A \ (Ax) \qquad \frac{\Gamma \vdash A \quad A, \Delta \vdash B}{\Gamma, \Delta \vdash B}\ (Cut)$$

$$\frac{\Gamma \vdash C}{\Gamma, !A \vdash C}\ (Weak.) \qquad \frac{\Gamma, !A, !A \vdash B}{\Gamma, !A \vdash B}\ (Contr.)$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B}\ (\multimap, r) \qquad \frac{\Gamma \vdash A \quad B, \Delta \vdash C}{\Gamma, A \multimap B, \Delta \vdash C}\ (\multimap, l)$$

$$\frac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C}\ (\otimes_i, l) \qquad \frac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B}\ (\otimes, r)$$

$$\frac{A_1, \ldots, A_n \vdash B}{!A_1, \ldots, !A_n \vdash !B}\ (!)$$

$$\frac{\Gamma, A \vdash B}{\Gamma, !A \vdash B}\ (\epsilon) \qquad \frac{\Gamma, !!A \vdash B}{\Gamma, !A \vdash B}\ (\delta)$$

$$\frac{\Gamma, T[S/t] \vdash C}{\Gamma, \forall t.T \vdash C}\ (\forall, l) \qquad \frac{\Gamma \vdash C}{\Gamma \vdash \forall t.C}\ t \notin FV(\Gamma)\ (\forall, r)$$

Expressivity of IMELL

- Intuitionistic logic (IL) can be interpreted inside Linear Logic with exponentials (LL)
- $(\_)^* : IL \rightarrow LL$
- $\Gamma \vdash_{IL} A$ iff $!\Gamma^* \vdash_{LL} A^*$
- It is actually a map on proofs
- Several interpretations have been studied, to establish properties also on their computational properties (i.e., under normalization/cut-elimination)
- Therefore: from our point of view LL is still way too expressive!

Fine control of duplication

- How are we allowed to use the duplicated resources (i.e., !-marked formulas)?
- Look at the various rules!
- Write $A \equiv B$ for $A \multimap B$ and $B \multimap A$
- The most fundamental property is $!A \equiv !A \otimes !A$
- It is obtained from rules (C), (W) and $(!)$ (check it!)
- But in LL (in order to interpret IL) we have more properties...
- $!A \multimap A$, from (ϵ) ("dereliction")
- $!A \multimap !!A$, from (δ) ("digging")
- The interplay between these rules is the main source for complexity of normalization and expressivity
- From a modal logic perspective: ! in LL is like $\Box$ in modal logic S4...

Subsystems of Linear Logic

	$!A \multimap !!A$	$!A \multimap A$	$!A \cong !A \otimes !A$
ELL	NO	NO	YES
LLL	NO	NO	YES
SLL	NO	$!A \multimap A \otimes \ldots \otimes A$	

...and their expressive power

ELL	Elementary Time
LLL	Polynomial Time
SLL	Polynomial Time

Subsystems of Linear Logic, II

As rules:

	(!)	(δ)	(ϵ)	(C)	(mplex)	(u!)
ELL	YES	NO	NO	YES	NO	derivable
LLL	NO	NO	NO	YES	NO	YES + (§)
SLL	YES	NO	NO	NO	YES	derivable

where

$$\frac{\Gamma, !A, !A \vdash B}{\Gamma, !A \vdash B}\ (C) \qquad \frac{A_1, \ldots, A_n \vdash B}{!A_1, \ldots, !A_n \vdash !B}\ (!)$$

$$\frac{\Gamma, A \vdash B}{\Gamma, !A \vdash B}\ (\epsilon) \qquad \frac{\Gamma, !!A \vdash B}{\Gamma, !A \vdash B}\ (\delta)$$

$$\frac{\Gamma, A, A] \vdash C}{\Gamma, !A \vdash C}\ (mplex) \qquad \frac{A \vdash C}{!A \vdash !C}\ u!$$

Subsystems of Linear Logic, III

- ELL has an elementary time cut-elimination procedure and represents (all) the elementary time functions.
  - Recall: elementary means to belong to $\mathcal{E}_3$ in Grzegorczyk hierarchy;
  - we have all the fixed-height towers of exponentials, but not the variable-height one
- SLL and LLL have a polytime cut-elimination procedure and represents (all) the polytime computable functions.
- We will consider (technically easier) "affine" variants of this logics, that is systems where full weakening is allowed.

We proceed in this way

- We introduce annotated sequent calculus of EAL/LAL ("A" stands for "affine")
- We argue (well: we just state) that the normal form of these lambda terms can be computed by considering their associated proof nets as intermediate calculus.
- In this intermediate calculus there are certain parameters of the nets that can be used to express the cost of normalization.

Elementary Affine Logic as an annotated sequent calculus

$$x : A \vdash x : A \ (Ax)$$

$$\frac{\Gamma \vdash M : A \quad x : A, \Delta \vdash N : B}{\Gamma, \Delta \vdash N[M/x] : B} \ (Cut)$$

$$\frac{\Gamma \vdash M : C}{\Gamma, x : A \vdash M : C} \ (Weak.)$$

$$\frac{\Gamma, x : !A, x : !A \vdash M : B}{\Gamma, x : !A \vdash M : B} \ (Contr.)$$

$$\frac{\Gamma \vdash N : A \quad x : B, \Delta \vdash M : C}{\Gamma, f : A \multimap B, \Delta \vdash M[(f\ N)/x] : C} \ (\multimap, l)$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \multimap B} \ (\multimap, r)$$

$$\frac{x_1 : A_1, \ldots, x_n : A_n \vdash M : B}{x_1 : !A_1, \ldots, x_n : !A_n \vdash M : !B} \ (!)$$

$$\frac{\Gamma, x : T[S/t] \vdash M : C}{\Gamma, x : \forall t.T \vdash M : C} \ (\forall, l)$$

$$\frac{\Gamma \vdash M : \forall C}{\Gamma \vdash M : \forall t.C} \ t \notin FV(\Gamma) \ (\forall, r)$$

Data types in EAL

- Data types can be defined as in System F, but with some "!" in the middle, to mark "reuse"
- Natural numbers (unary notation)
  $N \equiv \forall t.!(t \multimap t) \multimap !(t \multimap t)$
- Binary words
  $\mathbb{B} = \forall t.!(t \multimap t) \multimap !(t \multimap t) \multimap !(t \multimap t)$
- Operations on such data also get some "!" in their types
  For instance, on Church numerals:
  - Multiplication: $mul \equiv \lambda n.\lambda m.\lambda f.n(m\ f) : N \multimap N \multimap N$;
  - Squaring: $sqr \equiv \lambda n.mul\ n\ n : !N \multimap !N$
- These additional !s make it difficult to program in these systems. . .

Proof nets for EAL

- EAL-typed λ-calculus is not too well behaved. Even preservation of typing under reduction ("subject reduction") fails, in general.
- The real machine model to be used are proof nets
- Proof nets for EAL are the same as for LL, but with less normalization rules, because EAL have less rules concerning !
- Crucial points:
  - For any arc e in a proof-net, let d_e be the number of boxes containing e (this is the depth of the arc.)
  - For any proof net Π, let d_Π, be the maximum of all the d_e's, for e varying on all the arcs (this is the depth of the proof net.)
  - During reduction, the depth of any arc do not changes. This is specific to EAL. It is false for LL: dereliction (ϵ) will make it decrease; digging (δ) will make it increase.

Simulation lemma

To be more specific, proof nets can be used as an intermediate language in view of the following result:

Lemma

Let $\Gamma \vdash M : A$ and let Π_M the proof net associated to this proof. Now let $\Pi_M \to \Pi'$ in normal form. Then Π' corresponds to a proof of $\Gamma \vdash M' : A$, with $M \to M'$ and M' in normal form.

That is, normalization (i.e., computation) on proof nets, simulates normalization of the λ-term.

Complexity bounds for EAL

Theorem

Let Π be a proof net of depth d_Π. Then Π can be reduced to normal form in less than $2^{\cdot^{\cdot^{|\Pi|}}}\Big\}{d_\Pi \text{ times}}$.

Theorem

Let f be any elementary function (that is, $f \in \mathcal{E}_3$). Then there is a λ-term typeable in EAL (with type $N \multimap !^k N$) defining f.

Getting Light Affine Logic from EAL

- Take out the rule

$$\frac{x_1 : A_1, \ldots, x_n : A_n \vdash M : B}{x_1 : !A_1, \ldots, x_n : !A_n \vdash M : !B} \ (!)$$

- Instead, add its restricted version

$$\frac{x : A \vdash M : B}{x : !A \vdash M : !B} \ (u!)$$

(the rule may be applied also without environment $x : A$).

- To compensate for the loss, add a new modality, §, with rule

$$\frac{x_1 : A_1, \ldots, x_n : A_n, y : C_1, \ldots, y : C_m \vdash M : B}{x_1 : !A_1, \ldots, x_n : !A_n, y : \S C_1, \ldots, y : \S C_m \vdash M : \S B} \ (\S)$$

Data types in Light Affine Logic

- Data types can be defined as in EAL and System F, but with some "!" and § in the middle
- Natural numbers (unary notation)
  $N \equiv \forall t.!(t \multimap t) \multimap \S(t \multimap t)$
- Binary words
  $\mathbb{B} = \forall t.!(t \multimap t) \multimap !(t \multimap t) \multimap \S(t \multimap t)$
- Operations on such data also get some "!" and some § in their types
  For instance, on Church numerals:
  - Addition gets type $N \multimap N \multimap N$
  - Multiplication gets type $!N \multimap N \multimap \S N$
- These additional modalities make it difficult to compose and iterate on these terms.

Complexity bounds for LAL

As for EAL, the actual computational engine are the proof nets. This is required in order to get the polynomial bound.

Theorem

Let Π be a LAL proof net of depth d. Then Π can be reduced to normal form in less than $O((d+1)\cdot|\Pi|^{2^{d+1}})$

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

Theorem

Let f be any polytime computable function. Then there is a λ-term typeable in LAL (with type $\mathbb{B} \multimap \S^k\mathbb{B}$) defining f.

From Logic to Programming Languages

- How can a host machine assure the amount of resource needed to run a mobile program? A resource-aware type system or program-logic would provide implicit and verifiable certificates.
- In the realm of (first-order) term-rewriting systems, techniques like **quasi interpretations** have been shown to be useful for inferring complexity properties of programs (Bonfante et al.).
- **Type-systems** derived from non-size increasing computations have been exploited in the context of mobile resource guarantees (Hofmann et al., Beringer et al.).
- Enforcing resource-awareness in programming languages is not an easy task. The additional control provided cannot come at the price of unacceptable restrictions to programs.

Inferring Linear Bounds on Heap Size – Hofmann & Jost

- **Language**: first-order functional programming language with recursion.
- **Type-system**: simple types, including lists, with resource annotations.
- Example: $x : L(B, 2), 3 \vdash e : L(B, 4), 5$
  means
  - if we evaluate e starting with x bound to a list $[u_1, \ldots, u_m]$,
  - and we have a free-list of at least $3 + 2m$ cells,
  - then the computation will not get stuck from insufficient memory availability;
  - moreover, if the result is a list $[v_1, \ldots, v_n]$, then at the end the free-list will have at least $5 + 4n$ cells.

Hofmann & Jost, II

- **Type-system**: Contraction can only be done splitting the corresponding resource annotations: for example, from

$$x : \mathrm{L(B,3)}, y : \mathrm{L(B,6)} \vdash e : C, 7$$

we can derive

$$z : \mathrm{L(B,9)} \vdash e\{z/x, z/y\} : C, 7$$

- **Decorations**: given a skeleton of a type derivation (types, but not resource annotations) for e,
a set of linear inequalities $\mathcal{L}(e)$ is derived.
Solutions to $\mathcal{L}(e)$ are in one-to-one correspondence with valid type derivations for e.

From Logic to Computational Complexity

- Programming languages can be designed so that functions computable by acceptable programs extensionally correspond to certain computation complexity classes.
- If the underlying programming language is reasonably abstract, the system is then a machine-free characterization of a complexity class and can be used to infer properties of that same class.
- If we want to infer properties of a complexity class from properties of a certain system (which exactly characterizes it), we should keep the system as simple as possible, without emphasizing issues such as programming flexibility.

Challenges

- The area of implicit computational complexity appears very fragmented, with many different proposals.
- It is very difficult to compare relative **intensional expressive power**.
- It is not usually the case a system can be **extended** with new features preserving its quantitative properties
- Defining just another characterization of polynomial time is not enough.
- Deep, **foundational results** are extremely needed.

That's it, folks