Logic and Implicit Computational Complexity

Simona Ronchi Della Rocca

Emerita Professor
Università degli Studi di Torino

12th Panhellenic Logic Symposium

Anogeia, Crete, June 26-30, 2019

Classical Complexity theory

- study of complexity classes and their relations
- define first a computational model and its associated cost model (for time, space,...). The standard choice:

  Turing machines (**TM**)

  - measure of time: number of elementary steps
  - measure of space: dimension of the tape
  - the **TM** M works in bound f iff for any input u, $M(u)$ terminates using less than $f(|u|)$ resources
  - example: **PTIME** (**FPTIME**) is the set of decision problems (functions) that can be solved (computed) in polynomial time

Problems

- the design of a program and the proof of its complexity are done in two different steps
- it is difficult to supply formal proofs of complexity, since the difficulty of dealing formally with **TM***s*

Implicit Computational Complexity (ICC)

- describes complexity classes without explicit reference to a machine model and its cost bound
- uses techniques and results from Mathematical Logic:
  - proof theory (Curry-Howard correspondence)
  - recursion theory (restriction of primitive recursion schema)
  - model theory (finite model theory)
- two aims:
  - **A1** to implicitly characterize complexity classes
  - **A2** to supply programming languages with a certified complexity bound

The proof-theory approach

Let C be a complexity class.

- The schema:
  - start from a given programming language L
  - supply a type discipline for L in such a way that:
    - (soundness) if a program is well typed then its complexity belongs to C
    - (completeness) all and only the functions belonging to C are computed by well typed programs
- The realization is made by exploiting the so called Curry-Howard isomorphism, a bridge between the world of Intuitionistic Logic and that of functional programming languages, connecting:
  - formulae and types
  - proofs and programs
  - cut-elimination steps and computational rules

An example

- Choosing:
  - language: λ-calculus (with some extensions)
  - types: inspired by Soft Linear Logic (one of the light versions of LL)

The results

characterizations of **PTIME**, **FPTIME**, **PSPACE** and **NP**

(by Marco Gaboardi, Jean Yves Marion, SRDR)

Linear Logic (**LL**): a resource sensitive logic

- Formulae denote resources, and there two kinds:
  - A (linear resource, can be used once)
  - $!A$ (can be used as many times we want, also 0)
- The intuitionistic implication $A \to B$ is decomposed in $!A \multimap B$

  ($\multimap$ is the linear implication)
- Equivalence $!!A = !A$

The rules of Intuitionistic Linear Logic (**ILL**) (Girard,1987)

$$\frac{}{A \vdash A}\ (Id) \qquad \frac{\Gamma \vdash A \qquad \Delta, A \vdash B}{\Gamma, \Delta \vdash B}\ (cut)$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B}\ (\multimap R) \qquad \frac{\Gamma \vdash A \qquad B, \Delta \vdash C}{A \multimap B, \Gamma, \Delta \vdash C}\ (\multimap L)$$

$$\frac{\Gamma \vdash A \quad \alpha \notin FV(\Gamma)}{\Gamma \vdash \forall \alpha.A}\ (\forall R) \qquad \frac{\Gamma, B[C/\alpha] \vdash A}{\Gamma, \forall \alpha.B \vdash A}\ (\forall L)$$

$$\frac{\Gamma, \overbrace{A, \ldots, A}^{n \text{ times}} \vdash C}{\Gamma, !A \vdash C}\ (mpx) \qquad \frac{\Gamma \vdash A}{!\Gamma \vdash !A}\ (sp) \qquad \frac{\Gamma, !!B \vdash A}{\Gamma, !B \vdash A}\ (digging)$$

where n is the *rank* of the multiplexor (mpx) rule, Γ is a multiset of formulae, and Γ, Δ denotes multiset union.

Soft Linear Logic (**SLL**) (Lafont, 2004)

$$\frac{}{A \vdash A}\ (Id) \qquad \frac{\Gamma \vdash A \qquad \Delta, A \vdash B}{\Gamma, \Delta \vdash B}\ (cut)$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B}\ (\multimap R) \qquad \frac{\Gamma \vdash A \qquad B, \Delta \vdash C}{A \multimap B, \Gamma, \Delta \vdash C}\ (\multimap L)$$

$$\frac{\Gamma \vdash A \quad \alpha \notin FV(\Gamma)}{\Gamma \vdash \forall \alpha.A}\ (\forall R) \qquad \frac{\Gamma, B[C/\alpha] \vdash A}{\Gamma, \forall \alpha.B \vdash A}\ (\forall L)$$

$$\frac{\Gamma, \overbrace{A, \ldots, A}^{n \text{ times}} \vdash C}{\Gamma, !A \vdash C}\ (mpx) \qquad \frac{\Gamma \vdash A}{!\Gamma \vdash !A}\ (sp)$$

all rules of **ILL** but

$$\frac{\Gamma, !!B \vdash A}{\Gamma, !B \vdash A}\ (digging)$$

LL and SLL

- In **LL** the modality ! is unbounded :

$$!A \subseteq \{\underbrace{A \otimes A \otimes ... \otimes A}_{n} \mid n \in \mathbb{N}\}$$

- In **SLL** the modality ! is bounded :

$$!A \subset_{fin} \{\underbrace{A \otimes A \otimes ... \otimes A}_{n} \mid n \in \mathbb{N}\}$$

- The maximum rank of a multiplexor in a proof is a bound of the possible duplications during the cut-elimination procedure, so a bound for its complexity, both in time and space.

Properties of **SLL**

PTIME Soundness

The cut elimination procedure applied to a proof Π of rank n requires a number of steps

$$\leq |\Pi| \times n^d$$

where:

- $|\Pi|$ is the size of Π
- n is the maximum rank of a multiplexor in Π
- d is the maximum number of nested applications of rule (sp) in Π (depth of the proof).

PTIME Completeness

Every **PTIME** Turing Machine can be encoded by a **SLL** proof, in such a way that data are encoded by proofs with depth 0.

λ-calculus: the paradigm for functional programming languages

syntax:

$$M ::= x \mid \lambda x.M \mid MM$$

where x ranges over a countable set of variables.

$\lambda xy.M$ stands for $\lambda x.(\lambda y.M))$

$\lambda x_1 ... x_n.M$ stands for $\lambda x_1.(\lambda x_2.(...(\lambda x_n.M)...))$

$M_1 M_2 ... M_n$ stands for $M_1(M_2(...M_n))$.

operational semantics:

the β-reduction is the contextual closure of the rule:

$$(\lambda x.M)N \underset{\beta}{\longrightarrow} M[N/x]$$

where $M[N/x]$ denotes the (capture free) replacement of all occurrences of x in M by N.

Computing with λ-calculus

coding of natural numbers:

$$\underline{n} = \lambda xy.\, \underbrace{x(....(x\, y))}_{n}$$

coding of binary words:

$$\underline{< i_1 ... i_n >} = \lambda x_0 x_1 y.x_{i_1}(x_{i_2}(....(x_{i_n} y))$$

where i_j ranges over $\{0, 1\}$.

Theorem

In the λ-calculus, all and only the computable functions can be coded.

Types

STA is a natural deduction style type assignment system inspired by **SLL**, but:

- Terms are built in a linear way, and (mpx) rule is used for controlling variable duplication
  (technically this is realized by using as types a subset of the **SLL** formulae such that:
  - $\forall$ is not allowed on modal formulae
  - $!$ is not allowed on the right of $\multimap$)
- weakening and axiom introduce not modal formulae.

Types are the following subset of **SLL** formulae:

$$\begin{array}{lll} A & ::= & \alpha \mid \sigma \multimap A \mid \forall \alpha . A \qquad \text{(linear types)} \\ \sigma & ::= & A \mid !\sigma \end{array}$$

Rules of **STA**

$$\frac{}{x : A \vdash x : A}\ (Ax) \qquad \frac{\Gamma \vdash M : \sigma}{\Gamma, x : A \vdash M : \sigma}\ (w)$$

$$\frac{\Gamma, x : \sigma \vdash M : A}{\Gamma \vdash \lambda x.M : \sigma \multimap A}\ (\multimap I) \qquad \frac{\Gamma \vdash M : \sigma \multimap A \qquad \Delta \vdash N : \sigma \quad \Gamma \# \Delta}{\Gamma, \Delta \vdash MN : A}\ (\multimap E)$$

$$\frac{\Gamma, x_1 : \sigma, \ldots, x_n : \sigma \vdash M : \tau}{\Gamma, x :!\sigma \vdash M[x/x_1, ..., x/x_n] : \tau}\ (mpx) \qquad \frac{\Gamma \vdash M : \sigma}{!\Gamma \vdash M :!\sigma}\ (sp)$$

$$\frac{\Gamma \vdash A \quad \alpha \notin FV(\Gamma)}{\Gamma \vdash M : \forall \alpha.A}\ (\forall I) \qquad \frac{\Gamma \vdash M : \forall \alpha.A}{\Gamma \vdash M : A[B/\alpha]}\ (\forall E)$$

where $\Gamma \# \Delta$ denotes that the two contexts have disjoint variables.

Properties of **STA**

Subject reduction

$\Gamma \vdash M : \sigma$ and $M \underset{\beta}{\longrightarrow} M'$ imply $\Gamma \vdash M' : \sigma$.

Strong normalization

$\Gamma \vdash M : \sigma$ implies M is strongly normalizing.

The key points

- In the untyped language, β-reduction is not a cost model!
  1. $(\lambda xy.yx)N \underset{\beta}{\longrightarrow} \lambda y.yN$ (1 replacement)
  2. $(\lambda xy.yxx)N \underset{\beta}{\longrightarrow} \lambda y.yNN$ (2 replacements)
  3. $(\lambda xy.yxxx)N \underset{\beta}{\longrightarrow} \lambda y.yNNN$ (3 replacements)
  4. .....
- In a well typed program, β-reduction can be used as a cost model.

Cost of a reduction

Let M be typed by a derivation Π, and let $M \underset{\beta}{\longrightarrow} N$. Then this β-reduction can be performed by a **TM** in a number of steps $O(\mid M \mid^{3(d(\Pi)+1)})$.

Example

The key point is that, to be well typed, a term needs to be built in a linear way, using then the rule (mpx) for unifying the variables.

Let N be a closed linear term.

1. $(\lambda xy.yx)N \underset{\beta}{\longrightarrow} \lambda y.yN$

$$\Pi \triangleright \quad \dfrac{\dfrac{x : A \vdash \lambda y.yx : (A \multimap B) \multimap B \quad \vdash N : A}{\vdash \lambda xy.yx : A \multimap (A \multimap B) \multimap B}\ (\multimap I)}{\vdash (\lambda xy.yx)N : (A \multimap B) \multimap B}\ (\multimap E)$$

No duplication. No multiplexor is needed.

2. $(\lambda xy.yxx)N \underset{\beta}{\longrightarrow} \lambda y.yNN$

$$\Pi \triangleright \quad \dfrac{\dfrac{\dfrac{x_1 : A, x_2 : A \vdash \lambda y.yx_1x_2 : (A \multimap A \multimap B) \multimap B}{x :!A \vdash \lambda y.yxx : (A \multimap A \multimap B) \multimap B}\ (mpx)}{\vdash \lambda xy.yxx :!A \multimap (A \multimap A \multimap B) \multimap B}\ (\multimap I) \qquad \dfrac{\vdash N : A}{\vdash N :!A}\ (sp)}{\vdash (\lambda xy.yxx)N : (A \multimap A \multimap B) \multimap B}\ (\multimap E)$$

One duplication. At least one multiplexor (and one promotion) is needed.

measures

- the rank of a proof is the maximum between the ranks of the applications of (mpx) rule in it
- the depth of a proof is the maximum nesting of the the applications of (sp) rule in it

Quantitative properties of STA

Theorem (Complexity bound)

$\Pi \triangleright \Gamma \vdash M : \sigma$ *implies* M *reduces to normal form in* n β*-reduction steps, where*

$$n \leq \mid M \mid^{d(\Pi)+1}$$

and this implies that it reduces in normal form on a ***TM*** *in time:*

$$\leq \mid M \mid^{3\times(d(\Pi)+1)}$$

(where $\mid M \mid$ *is the number of symbol of* M *and* $d(\Pi)$ *is the maximal nesting of* (sp) *rule applications in* Π*)*

Remark

M can be assigned an infinite number of types. Every typing for M gives an upper bound to its reduction time.

Coding data types

- truth values:

$$\underline{true} = \lambda xy.x : \mathbf{B} \qquad \underline{false} = \lambda xy.y : \mathbf{B}$$

- iterators:

$$\underline{n} = \lambda xy.\underbrace{x(...x(x\,y)))}_{n} : \mathbf{N}_i$$

$$\mathbf{N}_i = \forall\alpha.\underbrace{!...!}_{i}(\alpha \multimap \alpha) \multimap \alpha \multimap \alpha$$

- natural numbers (binary words):

$$\underline{[b_0, b_1, \ldots, b_n]} = \lambda cz.cb_0(\cdots(cb_nz)) : \mathbf{S}_i$$

$$\mathbf{S}_i = \forall\alpha.\underbrace{!...!}_{i}(\mathbf{B} \multimap \alpha \multimap \alpha) \multimap \alpha \multimap \alpha$$

where $\mathbf{B} = \forall\alpha.\alpha \multimap \alpha \multimap \alpha$, $i \in \mathbf{Nat}$

Remark

Every data type can be typed by a derivation of depth 0.

Coding functions

- Let $\pi : N^p \to B$ be a predicate of arity p.
  A closed term $M = \lambda x_1 ... x_p . P$ codes π iff:
  - $M\underline{n}_1 ... \underline{n}_p = \underline{\pi(n_1 ... n_p)}$
  - $\vdash M : S_{i_1} \multimap S_{i_2} \multimap ... \multimap S_{i_p} \multimap \mathbf{B}, (i_j \in \mathbf{Nat})$
- Let $\phi : N^p \to N$ be a function of arity p.
  A closed term $M = \lambda x_1 ... x_p . P$ codes ϕ iff:
  - $M\underline{n}_1 ... \underline{n}_p = \underline{\phi(n_1 ... n_p)}$
  - $\vdash M : S_{i_1} \multimap S_{i_2} \multimap ... \multimap S_{i_p} \multimap S, (i_j \in \mathbf{Nat})$

Polynomial soundness

Theorem (Polynomial soundness)

Let P be a term coding either a predicate $\pi : N^p \to B$ or a function $\phi : N^p \to N$. Then $P\underline{n}_1 ... \underline{n}_p$ reduces to normal form in a number of steps polynomial in $| n_1 + n_2 + ... + n_p |$.

Proof.

All n_i are typed by derivations of depth 0. Let P be typed by a derivation of depth d, so the derivation for $P\underline{n}_1 ... \underline{n}_p$ has depth d, and reduces to normal form in a number of steps which is $\leq | P\underline{n}_1 ... \underline{n}_p |^d$. Since the size of the program is a constant with respect to the computation, this time is polynomial in $| n_1 + n_2 + ... + n_p |$. □

PTIME and **FPTIME** completeness

Theorem (PTIME completeness)

*All decision problems that can be computed in polynomial time by a **TM** can be coded in **STA**.*

Theorem (FPTIME completeness)

*All functions that can be computed in polynomial time by a **TM** can be coded in **STA**.*

From **PTIME** to **PSPACE**

PSPACE is the set of problems that can be solved by a **TM** in polynomial space.

- Problems:
  1. How to measure the space of a reduction?
     (In the real programming languages substitution is not explicitly performed)
  2. How to transfer our tools from time to space?
- Solution:
  1. Use an (abstract) reduction machine for λ-calculus
     (performing linear substitution)
  2. Use the following equivalence:

$$\textbf{PSPACE} = \textbf{APTIME}$$

where **APTIME** is the set of problems solved by an Alternating Turing Machine (**ATM**)
(i.e., computation that repeatedly forks into subcomputations and whose result is obtained by a backward computation from all the subcomputations results.)

Extensions of terms and types

- Terms:

$$M ::= x \mid 0 \mid 1 \mid \lambda x.M \mid MM \mid \texttt{if } M \texttt{ then } M \texttt{ else } M$$

- Reduction rules:

$$(\lambda x.M)N \underset{\beta}{\longrightarrow} M[N/x]$$

$$\texttt{if } 0 \texttt{ then } M \texttt{ else } N \ \rightarrow_\delta M \qquad \texttt{if } 1 \texttt{ then } M \texttt{ else } N \ \rightarrow_\delta N$$

$\rightarrow^*_{\beta\delta}$ denotes the reflexive and transitive closure of $\rightarrow_{\beta\delta}$.

- Types:

$$A ::= \mathbf{B} \mid \alpha \mid \sigma \multimap A \mid \forall \alpha.A \quad \text{(Linear Types)}$$

$$\sigma ::= A \mid !\sigma$$

The type system **STAB**

$$\frac{}{x : A \vdash x : A}\ (Ax) \qquad \frac{}{\vdash 0 : \mathbf{B}}\ (\mathbf{B}_0 I) \qquad \frac{}{\vdash 1 : \mathbf{B}}\ (\mathbf{B}_1 I) \qquad \frac{\Gamma \vdash M : \sigma}{\Gamma, x : A \vdash M : \sigma}\ (w)$$

$$\frac{\Gamma, x : \sigma \vdash M : A}{\Gamma \vdash \lambda x.M : \sigma \multimap A}\ (\multimap I) \qquad \frac{\Gamma \vdash M : \sigma \multimap A \quad \Delta \vdash N : \sigma \quad \Gamma \# \Delta}{\Gamma, \Delta \vdash MN : A}\ (\multimap E)$$

$$\frac{\Gamma, x_1 : \sigma, \ldots, x_n : \sigma \vdash M : \mu}{\Gamma, x : !\sigma \vdash M[x/x_1, \cdots, x/x_n] : \mu}\ (mpx) \qquad \frac{\Gamma \vdash M : \sigma}{!\Gamma \vdash M : !\sigma}\ (sp)$$

$$\frac{\Gamma \vdash M : A \quad \alpha \notin \mathrm{FTV}(\Gamma)}{\Gamma \vdash M : \forall \alpha.A}\ (\forall I) \qquad \frac{\Gamma \vdash M : \forall \alpha.B}{\Gamma \vdash M : B[A/\alpha]}\ (\forall E)$$

$$\frac{\Gamma \vdash M : \mathbf{B} \quad \Gamma \vdash N_0 : \sigma \quad \Gamma \vdash N_1 : \sigma}{\Gamma \vdash \ \mathtt{if}\ M\ \mathtt{then}\ N_0\ \mathtt{else}\ N_1\ : \sigma}\ (\mathbf{B}E)$$

Properties of **STAB**

Subject reduction

$\Gamma \vdash M : \sigma$ and $M \underset{\beta}{\longrightarrow} M'$ imply $\Gamma \vdash M' : \sigma$.

Strong normalization

$\Gamma \vdash M : \sigma$ implies M is strongly normalizing.

A leftmost outermost reduction machine

The machine is a set of rules of the shape:

$$\mathcal{C}, \mathcal{A} \models N \Downarrow b$$

where:

- $\mathcal{A}$ is the store, which allows to perform substitutions one occurrence at a time:

$$\mathcal{A} ::= \emptyset \mid \mathcal{A}@\{x := M\}$$

- $\mathcal{C}$ is a context remembering the computation path, which allows avoidance of backtracking:

$$C[\circ] ::= \circ \mid (\ \texttt{if}\ C[\circ]\ \texttt{then}\ L\ \texttt{else}\ R\ )V_1 \cdots V_n$$

- N is a program (a closed term of type $\mathbf{B}$)

The rules of the machine

$$\frac{}{C, \mathcal{A} \models \mathtt{b} \Downarrow \mathtt{b}} \ (Ax)$$

$$\frac{C, \mathcal{A}@\{x' := N\} \models M[x'/x]V_1 \cdots V_m \Downarrow \mathtt{b}^*}{C, \mathcal{A} \models (\lambda x.M)NV_1 \cdots V_m \Downarrow \mathtt{b}} \ (\beta)$$

$$\frac{\{x := N\} \in \mathcal{A} \quad C, \mathcal{A} \models NV_1 \cdots V_m \Downarrow \mathtt{b}}{C, \mathcal{A} \models xV_1 \cdots V_m \Downarrow \mathtt{b}} \ (h)$$

$$\frac{C[(\ \mathtt{if}\ [\circ]\ \mathtt{then}\ N_{\mathtt{0}}\ \mathtt{else}\ N_1\ )V_1 \cdots V_m], \mathcal{A} \models M \Downarrow \mathtt{0} \quad C, \mathcal{A} \models N_{\mathtt{0}}V_1 \cdots V_m \Downarrow \mathtt{b}}{C, \mathcal{A} \models (\ \mathtt{if}\ M\ \mathtt{then}\ N_{\mathtt{0}}\ \mathtt{else}\ N_1\ )V_1 \cdots V_m \Downarrow \mathtt{b}} \ (\ \mathtt{if}\ \mathtt{0})$$

$$\frac{C[(\ \mathtt{if}\ [\circ]\ \mathtt{then}\ N_{\mathtt{0}}\ \mathtt{else}\ N_1\ )V_1 \cdots V_m], \mathcal{A} \models M \Downarrow \mathtt{1} \quad C, \mathcal{A} \models N_1V_1 \cdots V_m \Downarrow \mathtt{b}}{C, \mathcal{A} \models (\ \mathtt{if}\ M\ \mathtt{then}\ N_{\mathtt{0}}\ \mathtt{else}\ N_1\ )V_1 \cdots V_m \Downarrow \mathtt{b}} \ (\ \mathtt{if}\ \mathtt{1})$$

(*) x' is a fresh variable.

PSPACE results

Let the abstract machine compute: $\mathcal{C}, \mathcal{A} \models M \Downarrow \mathbf{b}$. Then the space used by the machine during this computation is:

the maximal size of the store in the computation

+

the maximal size of the context in the computation

Theorem (Polynomial Space Soundness)

Let M be a program (a closed term of type $\mathbf{B}$), and let Π be a derivation of $\vdash M : \mathbf{B}$, and let $d(\Pi)$ be the depth of Π (the maximal nesting of applications of (sp) rule in Π). Then M reduces to normal form (through the given abstract machine) in space

$$\leq 3\times \mid M \mid^{3\times d(\Pi)+4}$$

Theorem (Polynomial Space Completeness)

*Every decision problem $\mathcal{D} \in$ **PSPACE** can be solved by a program in **STAB**.*

From **PSPACE** to **NP**

NP is the set of problems that can be decided in polynomial time by a non-deterministic **TM**.

- The language:

$$M ::= x \mid MM \mid \lambda x.M \mid M + M$$

- The reduction rules:

$$(\lambda x.M)N \underset{\beta}{\longrightarrow} M[N/x] \qquad M + N \to_\gamma M \qquad M + N \to_\gamma N$$

Remark

The calculus is not confluent!

The type system **STA+**

$$\frac{}{x : A \vdash x : A}\ (Ax) \qquad \frac{\Gamma \vdash M : \sigma}{\Gamma, x : A \vdash M : \sigma}\ (w)$$

$$\frac{\Gamma, x : \sigma \vdash M : A}{\Gamma \vdash \lambda x.M : \sigma \multimap A}\ (\multimap I) \qquad \frac{\Gamma \vdash M : \sigma \multimap A \qquad \Delta \vdash N : A \quad \Gamma \# \Delta}{\Gamma, \Delta \vdash MN : A}\ (\multimap E)$$

$$\frac{\Gamma, x_1 : \sigma, \ldots, x_n : \sigma \vdash M : A}{\Gamma, x :!\sigma \vdash M[x/x_1, ..., x/x_n] : A}\ (mpx) \qquad \frac{\Gamma \vdash \sigma}{!\Gamma \vdash !\sigma}\ (sp)$$

$$\frac{\Gamma \vdash A \quad \alpha \notin FV(\Gamma)}{\Gamma \vdash M : \forall \alpha.A}\ (\forall I) \qquad \frac{\Gamma \vdash M : \forall \alpha.A}{\Gamma \vdash M : A[B/\alpha]}\ (\forall E)$$

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash N : A}{\Gamma \vdash M + N : A}\ (sum)$$

Properties of **STA+**

Theorem (Subject Reduction)

Let $\Gamma \vdash M : \sigma$ *and* $M \to_{\beta\gamma} N$. *Then* $\Gamma \vdash N : \sigma$.

Remark

Terms with an exponential time behaviour can be typed: E.g.:

$$M \equiv \underline{\mathbf{n}}((\lambda x.zx + zx)I)$$

where $\underline{n}$ is a Church numeral and I is the identity.

Let M' be the normal form of M.

- $M \underset{\beta}{\longrightarrow}^* N \to_{\gamma}^* M'$ in a number of steps exponential in n (β-reductions according to an innermost strategy)
- $M \to_{\beta\gamma}^* M'$ in a number of steps linear in n (a suitable outermost strategy)

A non-deterministic abstract machine

$$\frac{C[\lambda x.[\circ]], \mathcal{A} \models M \Downarrow N}{C, \mathcal{A} \models \lambda x.M \Downarrow \lambda x.N}\ (\lambda) \qquad \frac{C, \mathcal{A}@\{x' := P\} \models M[x'/x]V_1 \cdots V_m \Downarrow N \quad x' \text{ fresh}}{C, \mathcal{A} \models (\lambda x.M)PV_1 \cdots V_m \Downarrow N}\ (\beta)$$

$$\frac{(*)\quad C[x[\circ]V_2 \cdots V_m], \mathcal{A} \models V_1 \Downarrow N_1 \quad \cdots \quad C[xN_1 \cdots N_{m-1}[\circ]], \mathcal{A} \models V_m \Downarrow N_m}{C, \mathcal{A} \models xV_1 \cdots V_m \Downarrow xN_1 \cdots N_m}\ (h_1)$$

$$\frac{\{x := P\} \in \mathcal{A} \quad C, \mathcal{A} \models PV_1 \cdots V_m \Downarrow N}{C, \mathcal{A} \models xV_1 \cdots V_m \Downarrow N}\ (h)$$

$$\frac{C, \mathcal{A} \models M_0V_1 \cdots V_m \Downarrow N}{C, \mathcal{A} \models (M_0 + M_1)V_1 \cdots V_m \Downarrow N}\ (L) \qquad \frac{C, \mathcal{A} \models M_1V_1 \cdots V_m \Downarrow N}{C, \mathcal{A} \models (M_0 + M_1)V_1 \cdots V_m \Downarrow N}\ (R)$$

$$(*)\ \ x \notin dom(\mathcal{A})$$

$$C ::= [\circ] \mid \lambda x.C[\circ] \mid xM_1 \ldots M_i[\circ]M_{i+1} \ldots M_n \qquad (1 \leq i \leq n)$$

NPTIME characterization

Theorem (NPTIME soundness)

Let Π be a derivation proving $\Gamma \vdash M : \sigma$, for some Γ, σ. Then M reduces to each one of its normal forms (by the non-deterministic abstract machine) in time bounded by $O(|M|^{O(d(\Pi))})$.

Theorem (NPTIME completeness)

*Every decision problem $\mathcal{D}$ decidable by a non deterministic **TM** in polynomial time can be solved by a term typable in **STA+**.*

Final considerations

- Further implicit characterizations of complexity classes by proof-theoretical methodologies:
  - **PTIME** (by types based on Light Linear Logic) (Baillot, Terui, 2009)
  - **PTIME** (by intersection types) (De Benedetti, RDR, 2014)
  - **ELEMENTARY TIME** (by types based on Elementary Linear Logic) (Coppola, Dal Lago, RDR, 2008)
  - **Exponential hierarchy** (by types based on Light Linear Logic) (Baillot, De Benedetti, RDR, 2018)
- Other approaches to ICC:
  - restriction on Gôdel-Kleene primitive recursion (Bellantoni, Cook, 1992)
  - stratified types (Leivant, Marion, 2000)
  - term rewriting (Bonfante, Marion, Moyen, 2005)
  - .....

Have the ICC aims been reached?

A1 Implicit characterization of complexity classes

YES All the results I have shown prove that alternative approaches to complexity are possible.

BUT For the moment no new results w.r.t. the classical theory have been reached.

Have the ICC aims been reached?

A2 Design of programming languages with a certified complexity bound

YES Each of the characterization of **PTIME**, **PSPACE**, **NP**, supplied before are examples of programming languages where every program is certified to have a complexity bound.

BUT Programming is not easy in these languages: e.g., usual iteration constructs are not allowed, quite few algorithms can be coded.

Toward algorithmic completeness

Functional completeness

A programming language is functionally complete with respect to a complexity class C if, for every function in C, there is at least one algorithm computing it with complexity bound in C.

Algorithmic completeness

A programming language is algorithmically complete with respect to a complexity class C if every algorithm with complexity bound C can be coded in it.

A programming language characterizing a given complexity class C is by definition functionally complete w.r.t. C. The algorithmic completeness is in some cases unreachable.

New ICC directions

A3 to enrich the bounded programming languages in order to increase their expressivity.

A4 to adapt the technical tools developed for **ICC** to statically control the complexity of programs at compilation time.

Bibliography

- Girard, J.-Y., Linear Logic,*Theoretical Computer Science* 50, p. 1-102, Elsevier (1987)
- Girard, J.-Y.,Light Linear Logic, *Information and Computation* 143(2), p. 175-204, 1998
- Lafont Y., Soft Linear Logic and Polynomial Time, *Theoretical Computer Science* 318, p. 163-180, Elsevier (2004)
- Gaboardi M. and Ronchi Della Rocca S., A Soft Type Assignment System for λ-Calculus, CSL 07, *Lecture Notes in Computer Science*, 4646, p. 253–267, 2007, Springer.
- Gaboardi M. and Marion J.Y. and Ronchi Della Rocca S., An Implicit Characterization of PSPACE, *ACM Transactions on Computational Logic*, 13(2):18:1–18:36, 2012
- Gaboardi M. and Ronchi Della Rocca S., From Light Logics to Type Assignments: a case study, *Logic Journal of the IGPL* 17, p. 499 – 530, 2009.

THANK YOU FOR YOUR ATTENTION!