Linear Logic, Types and Complexity

Simona Ronchi Della Rocca

Università degli Studi di Torino

Scuola Estiva di Logica 2015

Gargnano, August 28 2015
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
  - 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 bound
the design of a program and the proof of its complexity are made in two different steps

it is difficult to supply formal proofs of complexity, since the difficulty of dealing formally with TMs
Implicit Computational Complexity (ICC)

- describes complexity classes without explicit reference to a machine model and its cost bound
- uses techniques and results from Mathematical Logics:
  - proof theory
  - recursion theory
  - model theory
- two aims:
  - to implicitly characterize complexity classes
  - to supply programming languages with a certified complexity bound
A case study

- the idea:
  - start from a given programming language $L$
  - supply a type discipline for $L$ in such a way that:
    - if a term (program) is well typed then its complexity belongs to a given complexity class $C$
    - all and only the functions belonging to $C$ are computed by well typed terms
- the realization:
  - language: $\lambda$-calculus (with some extension)
  - types: inspired from Soft Linear Logic (one of the light versions of LL)

The results
characterizations of $\text{PTIME}, \text{FPTIME}, \text{PSPACE}$ and $\text{NP}$
\[ M ::= x \mid \lambda x. M \mid MM \]

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 \( \beta \)-reduction is the contextual closure of the rule:

\[
(\lambda x. M)N \xrightarrow{\beta} M[N/x]
\]

where \( M[N/x] \) denotes the (capture free) replacement of all occurrences of \( x \) in \( M \) by \( N \).
Computing with $\lambda$-calculus

coding of natural numbers:

$$ n = \lambda xy. x(....(xy)) $$

coding of binary words:

$$ <i_1...i_n> = \lambda x_0x_1y.x_i(x_{i_2}(....(x_{i_n}y))) $$

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

Theorem

*In the $\lambda$-calculus, all and only the computable functions can be coded.*
Linear Logic: a resource sensitive logic

- Two kinds of formulae:
  - $A$ (linear resource, can be used once)
  - $!A$ (can be used as many times we want, also 0)

- The intuitionistic implication $A \rightarrow B$ is decomposed in $!A \multimap B$
  ($\multimap$ is the linear implication)

- Equivalence $!!A = !A$
The rules of Intuitionistic Linear Logic (ILL)

\[
\begin{align*}
& \frac{}{A \vdash A} \quad (Id) \\
& \frac{\Gamma \vdash A \quad \Delta, A \vdash B}{\Gamma, \Delta \vdash B} \quad (cut) \\
& \frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B} \quad (-\multimap R) \\
& \frac{\Gamma \vdash A \quad B, \Delta \vdash C}{A \multimap B, \Gamma, \Delta \vdash C} \quad (-\multimap L) \\
& \frac{\Gamma \vdash A \quad \alpha \notin FV(\Gamma)}{\Gamma \vdash \forall \alpha.A} \quad (\forall R) \\
& \frac{\Gamma, B[C/\alpha] \vdash A}{\Gamma, \forall \alpha.B \vdash A} \quad (\forall L) \\
& \frac{n \text{ times}}{\Gamma, A, \ldots, A \vdash C} \quad (mpx) \\
& \frac{\Gamma \vdash A}{!\Gamma \vdash A} \quad (sp) \\
& \frac{\Gamma, !!B \vdash A}{\Gamma, !B \vdash A} \quad (digging)
\end{align*}
\]

where $\Gamma$ is a multiset of formulae, and $\Gamma, \Delta$ denotes multiset union.
Soft Linear Logic (**SLL**)
The fact that the equation: 

\[ \textit{!A = !!A} \]

allows to use the modality $!$ for counting the number of duplications of proofs during the cut-elimination procedure.

In **SLL** all the polynomial computations can be coded.
**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 \(\rightarrow\).

- Weakening and axiom introduce not modal formulae.

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

\[
A ::= \alpha \mid !\sigma \rightarrow A \mid \forall \alpha.A \quad \text{(linear types)}
\]

\[
\sigma ::= A \mid !\sigma
\]
Rules of **STA**

\[
\frac{x : A \vdash x : A}{(Ax)} \quad \frac{\Gamma \vdash M : \sigma}{\Gamma, x : A \vdash M : \sigma} \quad (w)
\]

\[
\frac{\Gamma, x : \sigma \vdash M : A}{\Gamma \vdash \lambda x. M : \sigma \to A} \quad (\to I) \quad \frac{\Gamma \vdash M : \sigma \to A}{\Delta \vdash N : \sigma} \quad \Gamma \# \Delta \quad (\to E)
\]

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

\[
\frac{\Gamma \vdash A \quad \alpha \not\in FV(\Gamma)}{\Gamma \vdash M : \forall \alpha. A} \quad (\forall I) \quad \frac{\Gamma \vdash M : \forall \alpha. A}{\Gamma \vdash M : A[B/\alpha]} \quad (\forall E)
\]

where \(\Gamma \# \Delta\) denotes that the two contexts have disjoint variables.
Properties of **STA**

subject reduction
\[ \Gamma \vdash M : \sigma \text{ and } M \xrightarrow{\beta} M' \text{ imply } \Gamma \vdash M' : \sigma. \]

strong normalization
\[ \Gamma \vdash M : \sigma \text{ implies } M \text{ is strongly normalizing.} \]
Example

Let $A = D \multimap D$, $C = A \multimap A \multimap A \multimap B$.

\[
\begin{align*}
\Pi &\vdash y : C \quad \lambda x. y x x x x : A \multimap B \\
\Pi &\vdash z : D \quad z : D \\
\Phi &\vdash \text{Id} : A \\
\Pi &\vdash I : !A \\
\Pi &\vdash y : C \quad (\lambda x. y x x x) I : B
\end{align*}
\]

where $\Pi_3[(Ax)/\Phi]$ denotes $\Pi_3$, where all axiom with subject $x$ has been replaced by $\Phi$.

- the rule $(sp)$ is a witness that $I$ will be copied (a number of times corresponding to the rank of $\Pi_3$).

- the rank of $\Pi_3$ counts the number of copies of an argument replacing $x$. 

\[
\begin{align*}
\Pi_3 &\vdash y : C, x_1 : A, x_2 : A \multimap y x_1 y x_2 : A \multimap B \\
\Pi &\vdash y, x_1 : A, x_2 : A, x_3 : A \multimap y x_1 y x_2 y x_3 : B
\end{align*}
\]
In the example, \( I \) will be copied a number of times less than or equal to \( r^d \) (\( 2^2 = 4 \)).

**NOTE:** counting is possible since the equation \( !A = !!A \) is no more valid.
Quantitative properties of STA

Theorem (complexity bound)

\[ \Pi \vdash \Gamma \vdash M : \sigma \text{ implies } M \text{ reduces to normal form in } n \beta\text{-reduction steps,} \]

where

\[ n \leq |M|^{d(\Pi)+1} \]

and this implies that it reduces in normal form on a \( TM \) in time:

\[ \leq |M|^{3 \times (d(\Pi)+1)} \]

(where \( |M| \) is the number of symbol of \( M \) and \( d(\Pi) \) is the maximal nesting of \((sp)\) rule applications in \( \Pi \))

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:
  
  \( \text{true} = \lambda xy.x : B \quad \text{false} = \lambda xy.y : B \)

- Iterators:

  \( n = \lambda xy. x(...x(xy))) : N_i \)

  \( N_i = \forall \alpha. \ldots! (\alpha \to^i \alpha) \to \alpha \to \alpha \)

- Natural numbers (binary words):

  \( [b_0, b_1, \ldots, b_n] = \lambda cz. cb_0(\ldots (cb_n z)) : S_i \)

  \( S_i = \forall \alpha. \ldots! (B \to^i \alpha \to \alpha) \to \alpha \to \alpha \)

where \( B = \forall \alpha. \alpha \to^i \alpha \to \alpha, i \in \text{Nat} \)

**Remark**

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

Let $\pi : N^p \rightarrow B$ be a predicate of arity $p$.
A closed term $M = \lambda x_1...x_p. P$ codes $\pi$ iff:

- $Mn_1...n_p = \pi(n_1...n_p)$
- $\vdash M : S_{i_1} \multimap S_{i_2} \multimap ... \multimap S_{i_p} \multimap B$, $(i_j \in \text{Nat})$

Let $\phi : N^p \rightarrow N$ be a function of arity $p$.
A closed term $M = \lambda x_1...x_p. P$ codes $\phi$ iff:

- $Mn_1...n_p = \phi(n_1...n_p)$
- $\vdash M : S_{i_1} \multimap S_{i_2} \multimap ... \multimap S_{i_p} \multimap S$, $(i_j \in \text{Nat})$
Polynomial soundness

Theorem (polynomial soundness)

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

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 $P_{n_1\ldots n_p}$ has depth $d$, and reduces to normal form in a number of steps which is $\leq |P_{n_1\ldots n_p}|^d$. Since the size of the program is a constant with respect to the computation, this time is polynomial in $|n_1 + n_2 + \ldots + n_p|$.
Theorem (**PTIME** characterization)

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

Theorem (**FPTIME** characterization)

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 $\lambda$-calculus  
     (performing linear substitution)
  2. Use the following equivalence:

\[
PSPACE = APTIME
\]

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

Terms:

\[ M ::= x \mid 0 \mid 1 \mid \lambda x.M \mid MM \mid \text{if } M \text{ then } M \text{ else } M \]

Reduction rules:

\[ (\lambda x.M)N \rightarrow_{\beta} M[N/x] \]

\[ \text{if } 0 \text{ then } M \text{ else } N \rightarrow_{\delta} M \quad \text{if } 1 \text{ then } M \text{ else } N \rightarrow_{\delta} N \]

\[ \rightarrow_{\beta\delta}^{*} \text{ denotes the reflexive and transitive closure of } \rightarrow_{\beta\delta}. \]

Types:

\[ A ::= B \mid \alpha \mid \sigma \rightarrow A \mid \forall \alpha.A \quad \text{(Linear Types)} \]

\[ \sigma ::= A \mid !\sigma \]
The type system **STAB**

\[
\begin{align*}
\text{(Ax)} & \quad \frac{x : A \vdash x : A}{\vdash x : A} \\
\text{(B₀I)} & \quad \frac{\vdash 0 : B}{\vdash 1 : B} \\
\text{(B₁I)} & \quad \frac{\Gamma \vdash M : \sigma}{\Gamma \vdash M : \sigma}
\end{align*}
\]

\[
\begin{align*}
\Gamma, x : \sigma \vdash M : A & \quad (\neg I) \\
\Gamma \vdash \lambda x. M : \sigma \neg A & \\
\Gamma \vdash M : \sigma \neg A & \quad (\neg E) \\
\Gamma, \Delta \vdash N : \sigma & \quad \text{Γ#Δ}
\end{align*}
\]

\[
\begin{align*}
\Gamma, x_1 : \sigma, \ldots, x_n : \sigma \vdash M : \mu & \quad (mpx) \\
\Gamma, x : \sigma \vdash M[x/x_1, \ldots, x/x_n] : \mu & \\
\neg \Gamma, M : \sigma & \quad (sp)
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash M : A & \quad \alpha \notin \text{FTV}(\Gamma) \\
\Gamma \vdash M : \forall \alpha. A & \quad (\forall I) \\
\Gamma \vdash M : \forall \alpha. B & \quad (\forall E)
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash M : B & \quad \Gamma \vdash N_0 : \sigma & \quad \Gamma \vdash N_1 : \sigma \\
\Gamma \vdash \text{if } M \text{ then } N_0 \text{ else } N_1 : \sigma & \quad \text{(BE)}
\end{align*}
\]
Properties of **STAB**

**subject reduction**

\[ \Gamma \vdash M : \sigma \text{ and } M \xrightarrow{\beta} M' \text{ imply } \Gamma \vdash M' : \sigma. \]

**strong normalization**

\[ \Gamma \vdash M : \sigma \text{ implies } M \text{ is strongly normalizing.} \]
A leftmost outermost reduction machine

The machine is a set of rules of the shape:

\[ C, A \vdash N \Downarrow b \]

where:

- \( A \) is the store, and it allows to perform substitutions one occurrence at a time:

\[ A ::= \emptyset | A@\{x := M\} \]

- \( C \) is a context remembering the computation path, and it allows to avoid backtracking:

\[ C[\circ] ::= \circ | ( \text{if} \ C[\circ] \text{ then } L \text{ else } R )V_1 \cdots V_n \]

- \( N \) is a program (a closed term of type \( B \))
The rules of the machine

\[
\begin{align*}
C, \mathcal{A} & \models b \downarrow b & (Ax) \\
C, \mathcal{A} \models (\lambda x. M)NV_1 \cdots V_m \downarrow b & \quad (\beta) \\
\{x := N\} \in \mathcal{A} & \quad C, \mathcal{A} \models NV_1 \cdots V_m \downarrow b & (h) \\
C((\text{if } [\circ] \text{ then } N_0 \text{ else } N_1)V_1 \cdots V_m], \mathcal{A} & \models M \downarrow 0 \quad C, \mathcal{A} \models N_0V_1 \cdots V_m \downarrow b & (\text{if } 0) \\
C((\text{if } [\circ] \text{ then } N_0 \text{ else } N_1)V_1 \cdots V_m], \mathcal{A} & \models M \downarrow 1 \quad C, \mathcal{A} \models N_1V_1 \cdots V_m \downarrow b & (\text{if } 1) \\
\end{align*}
\]

\( (*) \) \( x' \) is a fresh variable.
Example

Let $M = \lambda x. \text{if } x \text{ then } Ix \text{ else } O(Ix)1$ (where $I = \lambda x.x$ and $O = \lambda xy.y$).

Clearly $M0 \rightarrow^* 0$ and $M1 \rightarrow^* 1$ and both are programs.

Let us evaluate $M0$.

\[
\begin{align*}
\text{if } [\circ] \text{ then } Iy \text{ else } O(Iy)1, \{y := 0\} &\models 0 \downarrow \emptyset & (Ax) \\
\text{if } [\circ] \text{ then } Iy \text{ else } O(Iy)1, \{y := 0\} &\models y \downarrow \emptyset & (h) \\
\text{if } [\circ] \text{ then } Iy \text{ else } O(Iy)1, \{y := 0\} &\models y \downarrow \emptyset & (h) \\
\text{if } [\circ] \text{ then } Iy \text{ else } O(Iy)1, \{y := 0\} &\models Iy \downarrow \emptyset & (\beta) \\
\text{if } [\circ] \text{ then } Iy \text{ else } O(Iy)1 \downarrow \emptyset &\models 0 \downarrow \emptyset & (Ax) \\
\text{if } [\circ] \text{ then } Iy \text{ else } O(Iy)1 \downarrow \emptyset &\models z \downarrow \emptyset & (h) \\
\text{if } [\circ] \text{ then } Iy \text{ else } O(Iy)1 \downarrow \emptyset &\models Iy \downarrow \emptyset & (\beta) \\
\text{if } [\circ] \text{ then } Iy \text{ else } O(Iy)1 \downarrow \emptyset &\models 0 \downarrow \emptyset & (Ax) \\
\text{if } [\circ] \text{ then } Iy \text{ else } O(Iy)1 \downarrow \emptyset &\models 0 \downarrow \emptyset & (Ax)
\end{align*}
\]
PSPACE results

Let the abstract machine compute: \( C, \mathcal{A} \models M \Downarrow b \). 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 \( B \)), and let \( \Pi \) be a derivation of \( \vdash M : B \), and let \( d(\Pi) \) be the depth of \( \Pi \) (the maximal nesting of applications of \( sp \) rule in \( \Pi \)). Then \( M \) reduces to normal form (through the given abstract machine) in space

\[
\leq 3 \times |M|^{3 \times d(\Pi) + 4}
\]

Theorem (Polynomial Space Completeness)

Every decision problem \( \mathcal{D} \in PSPACE \) can be coded in \( STAB \).
From \textbf{PSPACE} to \textbf{NP}

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

- The language:

\[ M ::= x \mid MM \mid \lambda x. M \mid M + M \]

- The reduction rules:

\[
(\lambda x. M)N \xrightarrow{\beta} M[N/x] \quad M + N \rightarrow^\gamma M \quad M + N \rightarrow^\gamma N
\]

\textbf{Remark}

The calculus is not confluent!
The type system $\text{STA+}$

\[
\frac{x : A \vdash x : A}{(Ax)} \quad \frac{\Gamma \vdash M : \sigma}{\Gamma, x : A \vdash M : \sigma} \quad (w)
\]

\[
\frac{\Gamma, x : \sigma \vdash M : A}{\Gamma \vdash \lambda x. M : \sigma \rightarrow A} \quad (\sim I) \quad \frac{\Gamma \vdash M : \sigma \rightarrow A}{\Gamma, \Delta \vdash MN : A} \quad (\sim E)
\]

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

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

\[
\frac{\Gamma \vdash M : A \quad \Gamma \vdash N : A}{\Gamma \vdash M + N : A} \quad (sum)
\]
Properties of **STA**+

**Theorem (Subject Reduction)**

Let $\Gamma \vdash M : \sigma$ and $M \rightarrow_{\beta y} N$. Then $\Gamma \vdash N : \sigma$.

**remark**

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

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

where $n$ is a Church numeral and $I$ is the identity.

Let $M'$ be the normal form of $M$.

- $M \rightarrow_{\beta}^* N \rightarrow_{\gamma}^* M'$ in a number of steps exponential in $n$ ($\beta$-reductions according to an innermost strategy)

- $M \rightarrow_{\beta y}^* M'$ in a number of steps linear in $n$ (a suitable outermost strategy)
A non-deterministic abstract machine

\[
\begin{align*}
C[\lambda x.[\circ]], \mathcal{A} \models M \downarrow N \quad (\lambda) & \quad C, \mathcal{A} @ \{x' := P\} \models M[x'/x] V_1 \cdots V_m \downarrow N \quad x' \text{ fresh} \quad (\beta) \\
C, \mathcal{A} \models \lambda x. M \downarrow \lambda x. N & \\
\{x := P\} \in \mathcal{A} \quad C, \mathcal{A} \models PV_1 \cdots V_m \downarrow N \quad (h) \\
C, \mathcal{A} \models xV_1 \cdots V_m \downarrow N & \\
C, \mathcal{A} \models M_0 V_1 \cdots V_m \downarrow N \quad (L) & \quad C, \mathcal{A} \models M_1 V_1 \cdots V_m \downarrow N \quad (R) \\
C, \mathcal{A} \models (M_0 + M_1) V_1 \cdots V_m \downarrow N & \\
C ::= [\circ] | \lambda x. C[\circ] | xM_1 \cdots M_i[\circ] M_{i+1} \cdots M_n \quad (1 \leq i \leq n) \quad (\ast) & \quad x \notin \text{dom}(\mathcal{A})
\end{align*}
\]
**NPTIME characterization**

Theorem (**NPTIME**) Soundness)

Let $\Pi$ be a derivation proving $\Gamma \vdash M : \sigma$, for some $\Gamma, \sigma$. 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 coded by a term typable in $STA+$.
Bibliography


