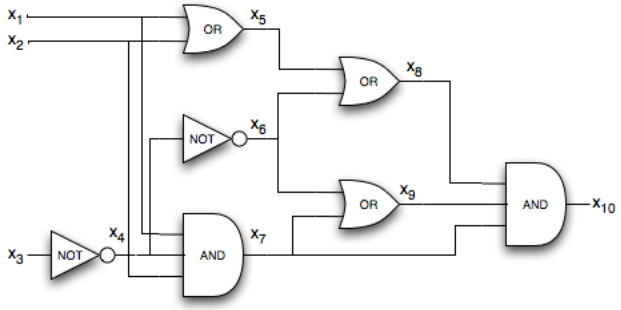
Non-uniformity in proofs and programs

Damiano Mazza
CNRS, LIPN, Université Paris 13

Journées PPS
Paris, 8-9 novembre 2018

Theoretical Computer Science =
Theory of Computation + Theory of Programming



Non-uniform computation

- A wholly different program for each input size...

$$F : \prod_{n:\mathsf{Nat}} (\mathsf{Bool}^n \to \mathsf{Bool}) \quad \text{vs. } \mathsf{Str} \to \mathsf{Bool} \qquad x : \mathsf{Str} \mapsto F_{|x|}(x)$$

- ...or a single program with access to advice:

$$(P, \alpha) : (\mathsf{Str} \times \mathsf{Str} \to \mathsf{Bool}) \times (\mathsf{Nat} \to \mathsf{Str}) \qquad x : \mathsf{Str} \mapsto P(\alpha(|x|), x)$$

- The two approaches are strongly related:
  - P/poly = polysize Boolean circuits;
  - L/poly = polysize branching programs.
- Interesting for practice (hardware) and theory (NP $\not\subseteq$ P/poly $\Rightarrow$ P $\neq$ NP)

Non-uniform computation in the λ-calculus

- Linear/affine λ-terms are "higher-order circuits":

$$\frac{\text{affine } \lambda\text{-terms}}{\lambda\text{-terms}} = \frac{\text{Boolean circuits}}{\text{Turing machines}}$$

- Consider affine terms of type $\mathsf{Bool}[A_1] \otimes \cdots \otimes \mathsf{Bool}[A_n] \multimap \mathsf{Bool}$.
  - HOPOLYSIZE $:=$ polysize affine terms;
  - $\mathsf{HOAC}^0 :=$ polysize terms of bounded height.

  Theorem (Terui, LICS 2004; M. and Terui, ICALP 2014).
  - HOPOLYSIZE $= \mathsf{P}/\mathsf{poly}$;
  - $\mathsf{HOAC}^0 = \mathsf{L}/\mathsf{poly}$.

What about advice?

- Non-uniformity exists in denotational semantics:
  - games semantics (Absramsky, Jagadeesan, Malacaria; Hyland, Ong; Nickau; Girard; Melliès...);
  - webbed models (Girard; Ehrhard, Regnier...)

$$\left[\!\!\left[\begin{array}{l} \lambda x : \mathsf{Bool}.\mathtt{if}\ x\ \mathtt{then} \\ \quad \mathtt{if}\ x\ \mathtt{then}\ A\ \mathtt{else}\ B \\ \mathtt{else} \\ \quad \mathtt{if}\ x\ \mathtt{then}\ C\ \mathtt{else}\ D \end{array} \right]\!\!\right] = \left\{ \begin{array}{l} ([\mathsf{tt},\mathsf{tt}], A),\ ([\mathsf{ff},\mathsf{ff}], D) \\ ([\mathsf{tt},\mathsf{ff}], B),\ ([\mathsf{ff},\mathsf{tt}], C) \end{array} \right\}$$

- Usually interpreted as non-determinism.
- In fact:
  1. we may see it as advice;
  2. the λ-calculus perspective allows us to link the family and advice approach via *continuous approximations*.

Girard's Approximation Theorem (1987), in higher dimension

$$!A = \lim_{n\to\infty} (A \,\&\, 1)^{\otimes n}$$

5. ...
4. residual equivalence (see Melliès's HDR)
3. standardization
2. cut-elimination/execution
1. proofs/programs
0. formulas/types

- Approximation order on affine reductions $\Longrightarrow$ poset on affine λ-terms.
- Ideals = infinitary programs $\Longrightarrow$ advice.

Approximations and continuity of reduction

$$M\ \underline{w} \longrightarrow \underline{b}$$

Approximations and continuity of reduction

$$
\begin{array}{c}
\lfloor M\ \underline{w} \rfloor_0 \\
\lfloor M\ \underline{w} \rfloor_1 \\
\vdots \\
\lfloor M\ \underline{w} \rfloor_m \\
\vdots \\
M\ \underline{w} \longrightarrow \underline{b}
\end{array}
$$

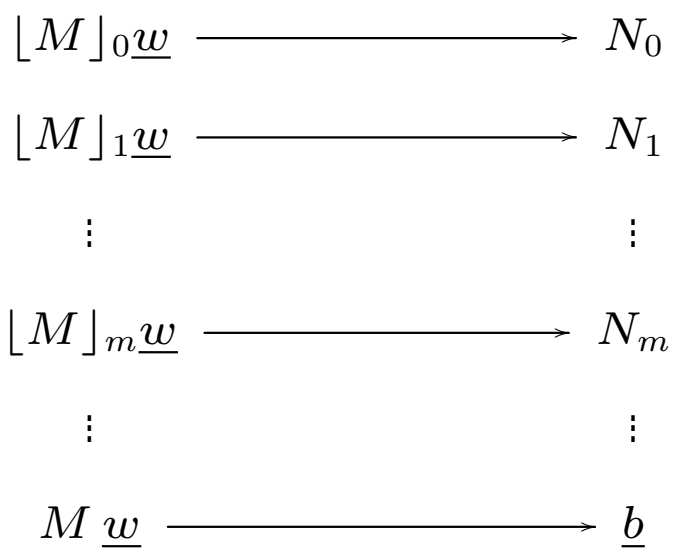
Approximations and continuity of reduction

$$
\begin{array}{c}
\lfloor M \rfloor_0 \lfloor \underline{w} \rfloor_0 \\
\lfloor M \rfloor_1 \lfloor \underline{w} \rfloor_1 \\
\vdots \\
\lfloor M \rfloor_m \lfloor \underline{w} \rfloor_m \\
\vdots \\
M\ \underline{w} \longrightarrow \underline{b}
\end{array}
$$

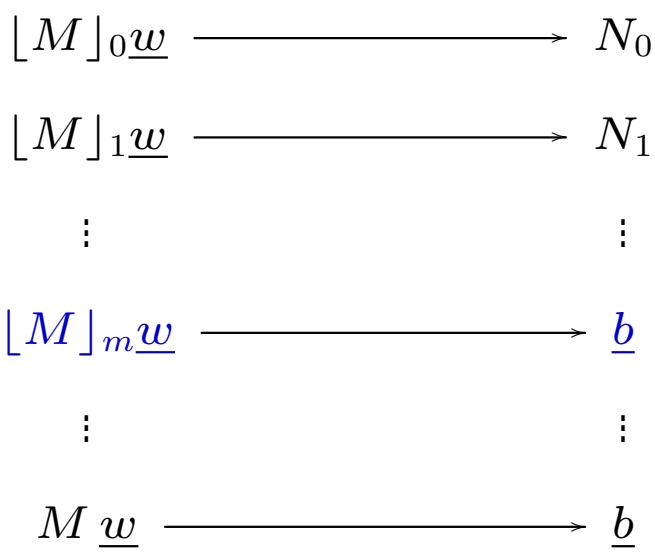
Approximations and continuity of reduction

$$
\begin{array}{c}
\lfloor M \rfloor_0 \underline{w} \\
\lfloor M \rfloor_1 \underline{w} \\
\vdots \\
\lfloor M \rfloor_m \underline{w} \\
\vdots \\
M\ \underline{w} \longrightarrow \underline{b}
\end{array}
$$

Approximations and continuity of reduction



Approximations and continuity of reduction



Approximations and continuity of reduction

$$
\begin{array}{ccc}
\lfloor M \rfloor_0 \underline{w} & \longrightarrow & N_0 \\
\lfloor M \rfloor_1 \underline{w} & \longrightarrow & N_1 \\
\vdots & & \vdots \\
\lfloor M \rfloor_m \underline{w} & \longrightarrow & \underline{b} \\
\vdots & & \vdots \\
M\ \underline{w} & \longrightarrow & \underline{b}
\end{array}
$$

- $|\lfloor M \rfloor_m| = O(\text{poly}(m))$ and $m = O(l(|w|))$ where $l(|w|) =$ # of steps.
- Polytime program (even infinitary!) admits polysize (higher order) circuits.
- Finite program $\Longrightarrow$ DLOGTIME-uniformity $\Longrightarrow$ Cook-Levin theorem.
- Algebraic uniformity: algorithmically uniform family vs. directed family.

Intersection types are approximations

- The archetipal example of non-uniform type system:

$$\dfrac{\dfrac{}{x : \alpha \to \beta \vdash x : \alpha \to \beta} \qquad \dfrac{}{x : \alpha \vdash x : \alpha}}{x : (\alpha \to \beta) \wedge \alpha \vdash xx : \beta}$$

- Intersection types arise from approximation presheaves:

$$\mathbf{L} \xrightarrow{\mathbf{G}} \Lambda_! \xrightarrow{\mathrm{Apx}[\mathcal{D}]} \mathfrak{Dist}$$

Grothendieck construction

$$\int(\mathrm{Apx}[\mathcal{D}] \circ \mathbf{G}) \xrightarrow{\mathbf{p}[\mathbf{D},\mathbf{G}]} \mathbf{L}$$

- Continuity is subject expansion $\Longrightarrow$ proof of Cook-Levin theorem via types.

Intermezzo: independence of P vs. NP and circuits

Lower bounds against non-uniform computation establish impossibility results for computation in the physical world: it could be that P $\neq$ NP*, yet* NP*-complete problems can still be efficiently solved using "bloated" programs with sufficiently many lines of code for large inputs. Non-uniform circuit size lower bounds for* NP *would rule out this possibility.*

Ryan Williams

- Formally, P $\neq$ NP is a Π_2^0 statement:

$$\forall M.\exists \varphi.\, (\mathrm{PolyTimeTM}(M) \Rightarrow \neg(\mathrm{Accept}(M,\varphi) \Leftrightarrow \textsc{Sat}(\varphi)))$$

- $\mathbf{T} + \Pi_1^0 \nvdash \forall M.\exists \varphi.A(M,\varphi)$ iff $\lambda M.\mu\varphi.A(M,\varphi)$ grows "too fast".
- Now, $\mathbf{T} \nvdash$ P $\neq$ NP proved with currently known method implies $\mathbf{T} + \Pi_1^0 \nvdash$ P $\neq$ NP, so there is $(M_n)_{n\in\mathbb{N}}$ s.t. M_n correctly decides SAT on formulas of size $\leq f(|M_n|)$, where f grows very quickly (accordingly to the power of $\mathbf{T}$).
- But a machine limited to bounded-size inputs is a cuircuit! So, for instance, $\mathbf{PRA} + \Pi_1^0 \nvdash$ P $\neq$ NP $\Longrightarrow$ SAT has circuits of size $O(n^{\mathrm{Ack}^{-1}(n)})$! (Kurtz, O'Donnell, Royer 1987).

The execution envelope of a program

- Let M decide a language: $M\underline{w} \to^* \underline{b_w}$. In affine intersection types:

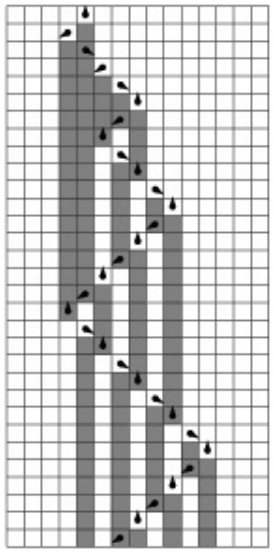
$$\vdash \underline{b_w} : \text{Bool} \xLeftarrow{\text{subj.exp.}} \vdash M\underline{w} : \text{Bool} \xLeftarrow{\text{synt.dir.}} \vdash M : \overset{\vdots\ \mathrm{env}(M)_w}{A} \to \text{Bool}$$

- The *running time* of M is the function

$$n \mapsto \max_{|w|=n} |\mathrm{env}(M)_w|$$

- The *execution space* of M is the function

$$n \mapsto \max_{|w|=n} \mathrm{depth}(\mathrm{env}(M)_w)$$



Computation + resources = non-uniform computation

Q: *Which results in complexity theory make essential use of uniformity?*

A: *I have asked many experts essentially this question, and the answer I always get is: none.*

TCS StackExchange

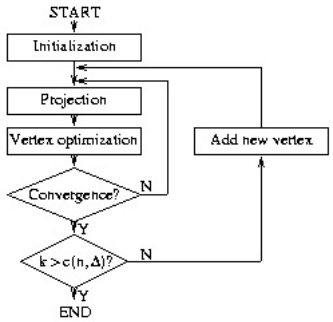
(L)es principaux résultats montrés en complexité, qui sont souvent énoncés dans le cadre de la classe P*, s'étendent immédiatement à* P/poly*, car on n'y utilise rien de spécifique sur les problèmes* P*, si ce n'est qu'ils ont les propriétés de nature combinatoire des problèmes* P/poly*!*

Bref, les vrais problèmes, en algorithmie se posent au niveau non uniforme (...). (Ce point de vue) conduira ceux d'entre vous qui ont une formation de logicien à des révisions déchirantes, comme de douter de la validité algorithmique de la notion même de fonction récursive. (...) (Un logicien) croit que les fonctions calculables en un sens plus réaliste constitueront nécessairement une sous-classe de celle des fonctions récursives. Eh bien, pas du tout: dans notre contexte, la mesure du temps et de la ressource éclipse complètement l'uniformité du calcul (...). (T)oute approche mathématique de la notion de calcul practicable n'est qu'une idéalisation, et ne doit être appréciée que pour ce qu'elle peut apporter comme réponses aux questions de l'algorithmie: de ce point de vue, la notion de fonction récursive (...) est remarquablement à coté de la plaque.

Bruno Poizat

The uselessness of the structure of algorithms?

- An algorithm usually has a structure, something we might want to call a "geometry".
- The theory and practice of programming are built around this structure: compositionnality, modularity, interfaces..., types, invariants, categorical semantics...
- However, resource analysis of structured programs is very hard.



- Execution envelope $\Longrightarrow$ "micro-programs" (circuits, etc.).
- Resource analysis of micro-programs is trivial, but global structure is lost (non-uniformity).
- Is there a fundamental incompatibility? (See also Harper 2014).

complexity of programs / structure of programs