

Implicit Computational Complexity

Simone Martini

Dipartimento di Scienze dell'Informazione
Università di Bologna
Italy

Scuola Aila 2009
Gargnano
24–29 agosto, 2009



Outline

Preliminaries

What is Implicit Computational Complexity

Complexity theory

Primitive recursion

Subrecursive classes

Grzegorzcyk hierarchy

Heinemann hierarchy

Bounded recursion on notation

Safe and tiered recursion

Bellantoni and Cook: Safe recursion

Leivant: Predicative (or tiered) recursion

Higher-order calculi: subsystems of Gödel's T

Proof Theory

Intuitionistic Logic and the Curry Howard Isomorphism



Implicit Computational Complexity

- ▶ *Standard* Computational Complexity
 - ▶ Study of **complexity classes** and their relations.
 - ▶ Define first a **machine model** and its associated **cost model(s)** (for time, space, etc.)
 - ▶ Define then complexity classes as **sets of problems** or functions, computable in a certain **bound**.
- ▶ *Implicit* Computational Complexity
 - ▶ Describe complexity classes **without** explicit reference to a machine model **and** to cost bounds.
 - ▶ It borrows techniques and results from Mathematical Logic
 - ▶ **Recursion Theory** (Restriction of primitive recursion schema);
 - ▶ **Proof Theory** (Curry-Howard correspondence);
 - ▶ **Model Theory** (Finite model theory).
 - ▶ It aims to define programming language tools (e.g., **type-systems**) enforcing resource bounds on the programs.



Complexity classes

- ▶ Standard machines: Turing automata.
 - ▶ Crucial: constant time elementary step.
 - ▶ Cost model: number of steps (time) or number of work cells (space).
 - ▶ TM M works in bound f iff for any input u , $M(u)$ terminates using less than $f(|u|)$ resources.
- ▶ Complexity classes
 - ▶ Sets of decision problems (functions with only 0 or 1 as values);
 - ▶ $\text{RESOURCE}[f(n)] = \{P \mid \text{there exists TM } M \text{ deciding } P \text{ and working in bound } f\}$;
- ▶ Some relevant classes
 - ▶ $\text{LOGSPACE} = \text{SPACE}[\log n]$;
 - ▶ $\text{LINTIME} = \text{TIME}[n]$;
 - ▶ $\text{PTIME} = \cup_{i \in \mathbb{N}} \text{TIME}[n^i]$;
 - ▶ $\text{PSPACE} = \cup_{i \in \mathbb{N}} \text{SPACE}[n^i]$;
 - ▶ $\text{EXPTIME} = \text{TIME}[2^n]$;



Invariance

- ▶ Classes are invariant w.r.t. linear factors:
 $\text{RESOURCE}[f(n)] = \text{RESOURCE}[af(n) + b];$
- ▶ Under certain assumptions, **different machine models** differ only by a **polynomial** in their use of resources.
E.g., if a problem P is solvable in bound f by a TM model, P is solved in at most f^k in another model.
- ▶ Therefore, under these assumptions, PTIME and PSPACE are very robust.



Coding of numbers

- ▶ Numbers must be coded into the TM alphabet.
- ▶ It is crucial that the coding of numbers be **positional with base greater than one**.
- ▶ With unary notation, the length of the input would be **exponentially longer** than the length in any other base. Therefore giving exponentially more resource to the computation. (Remember: the bound is a function of $|u|$).



Functional classes

- ▶ $\text{FP}_{\text{TIME}} =$
 $\{f : \mathbb{N} \rightarrow \mathbb{N} \mid$
there exists TM M computing f in polynomial bound};
- ▶ $\text{FLOGSPACE} = \dots;$
- ▶ \dots



Machine-free definitions of functions: Gödel-Kleene

Class of n -ary functions defined by **closure**.

▶ Base functions:

▶ Constant zero: $Z : \mathbb{N} \rightarrow \mathbb{N}$, $Z(y) = 0$;

▶ Successor: $S : \mathbb{N} \rightarrow \mathbb{N}$, $S(y) = y + 1$;

▶ Projections: for any $k \in \mathbb{N}$ and $i \leq k$, $\pi_i^k : \mathbb{N}^k \rightarrow \mathbb{N}$,
 $\pi_i^k(y_1, \dots, y_k) = y_i$.

▶ The function f is defined by **composition** from g, h_1, \dots, h_n if

$$f(y_1, \dots, y_k) = g(h_1(y_1, \dots, y_k), \dots, h_n(y_1, \dots, y_k))$$

▶ The function f is defined by **primitive recursion** from g and h if

$$\begin{aligned} f(0, \bar{y}) &= g(\bar{y}) \\ f(x + 1, \bar{y}) &= h(x, \bar{y}, f(x, \bar{y})) \end{aligned}$$



Classes of recursive functions

- ▶ The **primitive recursive functions** is the **least** class of functions containing the base functions and closed under composition and primitive recursion.
- ▶ The function f is defined by **minimization** from g if

$$f(\bar{y}) = \text{the least } z \text{ such that (i) } g(z, \bar{y}) = 0 \text{ and} \\ \text{(ii) } g(x, \bar{y}) \text{ is defined for all } x \leq z$$

Notation : $f(\bar{y}) = \mu z. g(z, \bar{y}) = 0$

- ▶ The **(general) recursive functions** is the least class of functions containing the base functions and closed under composition, (primitive recursion), and minimization.



Recursive functions as a machine model

- ▶ Original aim: define a class of functions *in extenso*.
- ▶ Natural operational interpretation as *rewriting*.
- ▶ However: no notion of *constant time* elementary step.
- ▶ Rewriting involves duplication of data of arbitrary size and of computations of arbitrary length.
- ▶ Need of non trivial data structures (stack) to (naïvely) implement primitive recursion.



Algebras for polynomial functions?

- ▶ We set out for a “closure-like” definition of FPTIME .
- ▶ We first study some known subclasses of the primitive recursive functions.



The spine of primitive recursion

$$f_0(x, y) = x + 1;$$

$$f_1(x, y) = x + y;$$

$$f_2(x, y) = xy;$$

$$f_{n+1}(x, 0) = 1;$$

$$f_{n+1}(x, y + 1) = f_n(x, f_{n+1}(x, y))$$

$$f_3(x, y) = x^y;$$

$$f_4(x, y) = x^{x^{x^{\dots^x}}} \Big\}^y \text{ times}.$$

Theorem

For any n and $x, y > 2$, $f_n(x, y) < f_{n+1}(x, y)$.



- ▶ Recursion causes bigger growth than composition:
 - ▶ Define $f^k(x) = (f \circ \dots \circ f)(x)$, k times.
 - ▶ For any n and any k , there exists \hat{x} such that, for any $x > \hat{x}$,
 $f_{n+1}(x, y) > f_n^k(x, x)$.
- ▶ The function f is defined by **bounded primitive recursion** from g, h and l iff f is defined by primitive recursion from g, h and moreover, for any \bar{x} ,

$$f(\bar{x}) < l(\bar{x}).$$

- ▶ For $n \geq 0$ the class \mathcal{E}_n is the least class including the base functions, the spine component f_n , and closed under composition and bounded primitive recursion.



Grzegorz hierarchy and complexity of computation

- ▶ The hierarchy is proper: $\mathcal{E}_n \subset \mathcal{E}_{n+1}$.
- ▶ Its limit are the primitive recursive functions: $\cup_n \mathcal{E}_n = \mathcal{PR}$.
- ▶ $f \in \mathcal{E}_n$ iff there exists a TM M computing f and a function $g \in \mathcal{E}_n$, such M works in time (space) bounded by g . (Unary notation used here).
- ▶ Hence the same holds for the primitive recursive functions.
- ▶ Do the classes \mathcal{E}_n correspond to natural complexity classes?

Theorem (RITCHIE, 1961)

$$\mathcal{E}_2 = \text{FLINSPACE}$$

- ▶ $\text{PTIME} \neq \text{FLINSPACE}$, but we do not know whether there is some inclusion between the two classes.



Many other hierarchies

- ▶ Many other hierarchies are definable, “structuring” recursion by levels.
- ▶ E.g., define the *rank* δ of a function definition:
 - ▶ Initial functions have rank 0;
 - ▶ f defined by composition from h, g_1, \dots, g_k have rank $\max\{\delta(h), \delta(g_1), \dots, \delta(g_k)\}$;
 - ▶ f defined by recursion from base g and step function h have rank $\max\{\delta(g), \delta(h) + 1\}$.
- ▶ $\mathcal{D}_n = \{f \mid \delta(f) \leq n\}$
- ▶ For $n \geq 2$, $\mathcal{D}_n = \mathcal{E}_{n+1}$ (Schwichtenberg; Müller, for $n = 2$).
- ▶ \mathcal{E}_3 is an important class: the **Kalmar elementary functions**.
- ▶ But we are mainly interested in the lower classes...



One last result for the “bigger” classes: PSPACE

PSPACE is the least class containing:

- ▶ Base functions: Zero, projections, max, $x^{|x|}$;
- ▶ Closed by composition, and
- ▶ Bounded primitive recursion.

Moral:

Bounded recursion, or just limiting nested recursion is **not** enough if we are interested in the lower complexity classes, e.g. PTIME. Indeed both PTIME and EXPTIME both lie in $\mathcal{D}_2 = \mathcal{E}_3$, that is the elementary functions.



A closer look: a notational problem

- ▶ Usual recursion—from $f(n)$ to $f(n + 1)$ —is **exponentially** long on the **size** of the input n .
- ▶ This is why controlling recursion, *per se*, is not enough:
 - ▶ A **single** recursion may cause exponential blow;
 - ▶ Two nested recursions are enough to reach the *elementary functions* (recall: $\mathcal{D}_2 = \mathcal{E}_3$).
- ▶ Move to **binary** representation for input (or, more generally, manipulate **strings**).



Recursion on Notation

- ▶ Data: binary strings
- ▶ Two “successors”:
 - ▶ s_0 , adding 0 at the least significant position
i.e., on the represented number $s_0(n) = 2n$;
 - ▶ s_1 , adding 1 at the least significant position
i.e., on the represented number $s_0(n) = 2n + 1$;
- ▶ Recursion on Notation:

$$f(0, \bar{y}) = g_0(\bar{y})$$

$$f(1, \bar{y}) = g_1(\bar{y})$$

$$f(s_0(x), \bar{y}) = h_0(x, \bar{y}, f(x, \bar{y}))$$

$$f(s_1(x), \bar{y}) = h_1(x, \bar{y}, f(x, \bar{y}))$$



Recursion on Notation, examples

- ▶ Now recursion converges quickly to a base case: $f(n)$ involves at most $\log n$ recursive calls.
- ▶ Notation: we mix strings and numbers.
- ▶ Example: duplicating the length of the input
As strings (\cdot is concatenation):

$$d(0) = d(1) = 1$$

$$d(s_0(x)) = d(x) \cdot 00$$

$$d(s_1(x)) = d(x) \cdot 00$$

As numbers ($*$ is multiplication):

$$d(0) = d(1) = 1$$

$$d(n) = 4 * d(\lfloor x/2 \rfloor)$$

That is, $d(n) = 2^{2|n|}$, that is $|d(n)| = 2|n| - 1$.



Recursion on notation is too generous

Recall

$$d(0) = d(1) = 1$$

$$d(s_0(x)) = d(x) \cdot 00$$

$$d(s_1(x)) = d(x) \cdot 00$$

And define

$$e(0) = e(1) = 1$$

$$e(s_0(x)) = d(e(x))$$

$$e(s_1(x)) = d(e(x))$$

Now $e(x)$ has **exponential** length in $|x| \dots$

Still too much growth...



Bounded recursion on notation

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

$$f(0, \bar{y}) = g_0(\bar{y})$$

$$f(1, \bar{y}) = g_1(\bar{y})$$

$$f(s_0(x), \bar{y}) = h_0(x, \bar{y}, f(x, \bar{y}))$$

$$f(s_1(x), \bar{y}) = h_1(x, \bar{y}, f(x, \bar{y}))$$

provided $f(x, \bar{y}) \leq k(x, \bar{y})$.



Cobham characterization of $FPTIME$

- ▶ However, the basic functions Zero, projections and successor do not grow enough. . .
- ▶ Let $x\#y = 2^{|\cdot| \cdot |\cdot|}$ (note: $|x|^k = |x|\# \dots \# |x|$).

Theorem (Cobham)

$FPTIME$ is the least class containing: Zero, the projections, the two successors on strings, $\#$; and closed under composition and bounded recursion on notation.

- ▶ Proof: $FPTIME \subseteq COB$: Code TMs as functions of the algebra. The iteration of the transition function is representable because *a priori* polynomially bounded.
 $COB \subseteq FPTIME$: By induction on the length of the definition, show that any function is computable by a polynomially bounded TM, exploiting the bound on the recursive definition.



Variations on a theme

- ▶ LOGSPACE is an important measure. LOGSPACE reductions are crucial to study the structure of PTIME, e.g. the existence of complete problems.
- ▶ A function $f : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ is defined by **strict** bounded recursion on notation from $g_0, g_1 : \mathbb{N}^n \rightarrow \mathbb{N}$, $h_0, h_1 : \mathbb{N}^{n+2} \rightarrow \mathbb{N}$ and $k : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ if

$$f(0, \bar{y}) = g_0(\bar{y})$$

$$f(1, \bar{y}) = g_1(\bar{y})$$

$$f(s_0(x), \bar{y}) = h_0(x, \bar{y}, f(x, \bar{y}))$$

$$f(s_1(x), \bar{y}) = h_1(x, \bar{y}, f(x, \bar{y}))$$

provided $f(x, \bar{y}) \leq |k(x, \bar{y})|$.



LOGSPACE

Theorem (Lind; Clote & Takeuti)

FLOGSPACE is the least class containing: Zero, projections, successors, length functions, bit selection, #; and closed under composition, strict bounded recursion on notation, and concatenation recursion on notation.

where *Concatenation Recursion on Notation* (CRN) from g, h_0, h_1 ($h_i(x, \bar{y}) \leq 1$) is

$$f(0, \bar{y}) = g_0(\bar{y})$$

$$f(1, \bar{y}) = g_1(\bar{y})$$

$$f(s_0(x), \bar{y}) = s_{h_0(x, \bar{y})}(f(x, \bar{y}))$$

$$f(s_1(x), \bar{y}) = s_{h_1(x, \bar{y})}(f(x, \bar{y}))$$



A critique on Cobham characterization

- ▶ Cobham's paper is the birth of computational complexity as a respected theory.
- ▶ It characterized P_{TIME} as a mathematically meaningful class.
- ▶ From the implicit computational complexity perspective, however. . .
 - ▶ It is not as implicit as it seems
 - ▶ It uses an explicit *a priori* bound on the construction
 - ▶ It “*throws in*” the polynomials (i.e., the $\#$ function) in the recipe, in order to make it work.
- ▶ We had to wait until the '80s to get a more “implicit” characterization of P_{TIME} . . .



Safe Recursion: idea

- ▶ Unbounded recursion schema to control the growth of functions
- ▶ Function arguments are partitioned into two separate classes.
- ▶ Function definitions are constrained to respect this partition.
- ▶ The arguments to a function $f : \mathbb{N}^n \rightarrow \mathbb{N}$ are partitioned into $m \leq n$ normal arguments and $n - m$ safe arguments:

$$f(x_1, \dots, x_m; x_{m+1}, \dots, x_n).$$

- ▶ Idea: calls to functions obtained by recursion can only appear in the safe zone.
- ▶ Need to modify the composition, in order to respect the distinction normal/safe.



Safe Recursion and Composition

- ▶ The function f is defined by **safe composition** from $g, h_1, \dots, h_n, k_1, \dots, k_m$ if

$$f(\bar{x}; \bar{y}) = g(h_1(\bar{x}; \bar{y}), \dots, h_n(\bar{x}; \bar{y}); k_1(\bar{x}; \bar{y}), \dots, k_m(\bar{x}; \bar{y})).$$

- ▶ The function f is defined by **safe recursion on notation** from g_0, g_1, h_0, h_1 if

$$f(0, \bar{x}; \bar{y}) = g_0(\bar{x}; \bar{y})$$

$$f(1, \bar{x}; \bar{y}) = g_1(\bar{x}; \bar{y})$$

$$f(s_0(x), \bar{x}; \bar{y}) = h_0(x, \bar{x}; \bar{y}, f(x, \bar{x}; \bar{y}))$$

$$f(s_1(x), \bar{x}; \bar{y}) = h_1(x, \bar{x}; \bar{y}, f(x, \bar{x}; \bar{y}))$$



Understanding safe composition and recursion

- ▶ The key clause:

$$f(s_i(x), \bar{x}; \bar{y}) = h_i(x, \bar{x}; \bar{y}, f(x, \bar{x}; \bar{y}))$$

- ▶ If f is defined by safe recursion:
 - ▶ it takes the recursion input $s_i(x)$ from the **normal** part;
 - ▶ but the recursive value $f(x, \bar{x}; \bar{y})$ is substituted into a **safe position** of h
 - ▶ then this recursive value will stay in a safe position, because of **safe composition**

$$f(\bar{x}; \bar{y}) = g(h_1(\bar{x};), \dots, h_n(\bar{x};); k_1(\bar{x}; \bar{y}), \dots, k_m(\bar{x}; \bar{y})).$$

and will not be copied back into a normal position.

- ▶ Intuitively, the depth of sub-recursions which h_i performs on y or \bar{y} cannot depend on the value being recursively computed.



Projections

- ▶ We have projections from both normal and safe zones

$$\pi_j^{n+m}(x_1, \dots, x_n; x_{n+1}, \dots, x_{n+m}) = x_j \quad 1 \leq j \leq n+m$$

- ▶ Now we can move arguments from safe to normal (but not vice-versa)
 - ▶ Assume we have $f(x; y, z)$.
 - ▶ Define $f'(x, y; z)$ same as f but with y “demoted” to normal
 - ▶ $f'(x, y; z) = f(\pi_1^2(x, y;); \pi_2^3(x, y; z), \pi_3^3(x, y; z))$



Controlling recursion by safeness

Successors are safe: $s_0(;x), s_1(;x)$

We have projections from both normal and safe zones

Recall the function

$$\begin{aligned}d(0) = d(1) &= 1 \\d(s_0(x)) = d(s_1(x)) &= d(x) \cdot 00\end{aligned}$$

Define:

$$\begin{aligned}d(0;) = d(1;) &= 1 \\d(s_0(x);) = d(s_1(x);) &= s_0(;s_0(;d(x;)))\end{aligned}$$

where formally the step function h is

$$h(x; z) = \pi_2^2(x; s_0(; s_0(; \pi_2^2(x; z))))$$



Controlling recursion by safeness, II

Recall now the **exponential function**

$$\begin{aligned}e(0) = e(1) &= 1 \\ e(s_0(x)) = e(s_1(x)) &= d(e(x))\end{aligned}$$

We cannot define e by safe recursion:

$$\begin{aligned}e(0;) = e(1;) &= 1 \\ e(s_0(x);) = e(s_1(x);) &= ? d(e(x)) ?\end{aligned}$$

The safe recursion schema requires $h(z; y) = d(; y)$,
but d is instead defined as $d(y;)$.



Polytime and safe recursion

Let \mathcal{B} be the function algebra containing

- ▶ successors: $s_0(;x), s_1(;x)$;
- ▶ projections, from normal and safe arguments;
- ▶ predecessor: $p(;0) = 0$ and $p(;s_i(x)) = x$;
- ▶ conditional:

$$C(;x, y, z) = \begin{cases} y & \text{if } x = s_0(v) \\ z & \text{if } x = s_1(v). \end{cases}$$

and closed under safe composition and recursion.

Theorem (Bellantoni and Cook)

*The polynomial time computable functions are exactly those functions of \mathcal{B} having only **normal** inputs.*



Proof of BC's theorem

- ▶ *Soundness: Any function in \mathcal{B} is polytime.*
 - ▶ Derive first a bound on the computed value: Let $f \in \mathcal{B}$. There is a polynomial q_f such that
$$|f(\bar{x}; \bar{y})| \leq q_f(|\bar{x}|) + \max(y_1, \dots, y_n)$$
 - ▶ Observe that such q_f 's are definable in Cobham's class.
 - ▶ Therefore, any instance of Safe recursion is an instance of Bounded rec. on notation.
- ▶ *Completeness: Any polytime function is in \mathcal{B} .*
 - ▶ Use Cobham characterization via bounded recursion on notation.
 - ▶ By induction on derivation on Cobham's system, show that for any polytime $f(\bar{y})$ there exists a function $f' \in \mathcal{B}$ and a polynomial p_f such that $f'(w; \bar{y}) = f(\bar{y})$, for all \bar{y} and all $w \geq p_f(|\bar{y}|)$
 - ▶ Now construct a function b in \mathcal{B} such that $b(\bar{x};) \geq p_f(|\bar{x}|)$
 - ▶ Set $f(\bar{x};) = f'(b(\bar{x};); \bar{x})$.



Variations: Safe *Affine* Composition

- ▶ In safe composition a safe argument may be used several times

$$f(\bar{x}; \bar{y}) = g(h_1(\bar{x}; \cdot), \dots, h_n(\bar{x}; \cdot); k_1(\bar{x}; \bar{y}), \dots, k_m(\bar{x}; \bar{y})).$$

- ▶ If we are interested in LOGSPACE, we must limit reuse of resources, imposing some kind of **lineary** constraint: any safe argument should be used at most once.
- ▶ The function f is defined by **safe affine composition** from $g, h_1, \dots, h_n, k_1, \dots, k_m$ if

$$f(\bar{x} : \bar{y}) = g(h_1(\bar{x} : \cdot), \dots, h_n(\bar{x} : \cdot) : k_1(\bar{x} : \bar{Y}_1), \dots, k_m(\bar{x} : \bar{Y}_m))$$

where any y_1, \dots, y_k of \bar{y} occurs at most once in any $\bar{Y}_1, \dots, \bar{Y}_m$.



Safe Affine Recursion: Logarithmic Space

- ▶ The function f is defined by **safe affine course-of-value recursion** on notation from g_0, g_1, h_0, h_1 if

$$f(0, \bar{x} : \bar{y}) = g_0(\bar{x} : \bar{y})$$

$$f(1, \bar{x} : \bar{y}) = g_1(\bar{x} : \bar{y})$$

$$f(s_0(x), \bar{x} : \bar{y}) = h_0(x, \bar{x} : f(x', \bar{x} : \bar{y}))$$

$$f(s_1(x), \bar{x} : \bar{y}) = h_1(x, \bar{x} : f(x'', \bar{x} : \bar{y})) \text{ with } x', x'' \leq x$$

Theorem (Mairson and Neergaard, 2003)

The set of logarithmic space functions equals the set of functions definable by safe affine course-of-value recursion, safe affine composition, and containing the base functions of BC.



Tiering

- ▶ Related to safe recursion is the notion of **predicative recurrence**, or tiering [Leivant, 1993].
- ▶ Any function and argument position comes with a *tier*.
- ▶ Equivalently: we have an infinite number of *copies* of the base data:

$$\mathbb{N}^0, \mathbb{N}^1, \mathbb{N}^2, \dots$$

- ▶ Functions have a type of the form
$$f : \mathbb{N}^i \times \dots \times \mathbb{N}^j \rightarrow \mathbb{N}^k$$
- ▶ Base functions are available at any tier.
- ▶ Composition is tier-preserving: $f^i \circ g^i = h^i$.



Predicative Recurrence - I

- ▶ Recursion is possible only over a variable with tier greater than that of the function:

$$f(0, y)^i = g_0(y^k)^i$$

$$f(1, y)^i = g_1(y^k)^i$$

$$f(s_0(x)^l, y)^i = h_0(x^l, y^k, f(x, y)^i)^i \text{ with } l > i$$

$$f(s_1(x)^l, y)^i = h_1(x^l, y^k, f(x, y)^i)^i \text{ with } l > i$$

- ▶ In other words:
 - ▶ When defining inductively
$$f(s_b(x), y) = h_b(x, y, f(x, y))$$
 - ▶ we must have
$$h_b : \mathbb{N}^l \times \mathbb{N} \times \mathbb{N}^i \rightarrow \mathbb{N}^i$$
with $l > i$, and we obtain
$$f : \mathbb{N}^l \times \mathbb{N} \rightarrow \mathbb{N}^i$$



Examples of predicative recurrence

Recall: In $f(s_b(x)^l, y)^i = h_b(x^l, y^k, f(x, y)^i)^i$, $l > i$.

- ▶ Flat recurrence: the stratification is vacuous, because the recursion argument is absent

$$\rho(s_b(x)) = x$$

- ▶ Concatenation:

$$\oplus(\epsilon, y) = y$$

$$\oplus(s_b(x), y) = s_b(\oplus(x, y))$$

Imposing stratification:

$$\oplus(s_b(x)^l, y^j)^i = s_b(\oplus(x^l, y^j)^i) \text{ with } l > i$$

Take $l = 1$, $i = 0$ (and j whatever, say 0):

$$\oplus : \mathbb{N}^1 \times \mathbb{N}^0 \rightarrow \mathbb{N}^0$$



Examples of predicative recurrence - II

We can apply predicative recurrence on any constructor algebra:
numbers in unary or binary notation, trees, etc.

- ▶ Addition in unary notation:

$$\begin{aligned}+(0, y) &= 0 \\+(s(x), y) &= s(+ (x, y))\end{aligned}$$

Imposing stratification:

$$\begin{aligned}+(s(x)^1, y^0)^1 &= s(+ (x^1, y^0)^1) \\+ : \mathbb{N}^1 \times \mathbb{N}^0 &\rightarrow \mathbb{N}^0\end{aligned}$$

- ▶ Multiplication in unary notation:

$$\begin{aligned}*(0, y) &= 0 *(s(x), y) &= + (y, *(x, y))\end{aligned}$$

Impose the stratification for $+$:

$$*(s(x), y) = + (y^1, *(x, y)^0)^0$$

and propagate; everything is OK: $* : \mathbb{N}^1 \times \mathbb{N}^1 \rightarrow \mathbb{N}^0$



A non predicative recurrence

Recall: In $f(s(x)^l, y)^i = h(x^l, y^k, f(x, y)^i)^i$, $l > i$.

- ▶ Powers of two $P2(n) = 2^n$:

$$P2(0) = 1$$

$$P2(s(x)) = +(P2(x), P2(x))$$

Recall that $+: \mathbb{N}^1 \times \mathbb{N}^0 \rightarrow \mathbb{N}^0$

and impose this stratification:

$$P2(s(x)^?)^{??} = +(P2(x)^1, P2(x)^0)^0$$

The first input to $+$ must have level greater than the output

From the output of $+$ we would get $?? = 0$

From the first input to $+$ we would get $?? = 1$.

Impossible under any assignment.



Predicative recurrence and polynomial time

Theorem (Leivant, 1993)

Let W be a free algebra, f a function over W . The following are equivalent:

- 1. f is computable in time polynomial in the maximal height of the inputs.*
- 2. f is definable by predicative recursion over A^0 and A^1 .*
- 3. f is definable by predicative recursion over arbitrary A^i 's, $i \geq 0$.*

Compare to Bellantoni and Cook: no initial functions.

Same idea...



Tiering and Safe recursion

- ▶ Tiering and safeness are equivalent
 - ▶ From a tiered $f(x_1^{l_1}, \dots, x_n^{l_n}, y_1^i, \dots, y_m^i)^i$ where $l_1, \dots, l_n > i$ we get $f(x_1, \dots, x_n; y_1, \dots, y_m)$
 - ▶ From a safe definition $f(x_1, \dots, x_n; y_1, \dots, y_m)$ for any tier i , there is a tiered definition of f in which $f(x_1^{l_1}, \dots, x_n^{l_n}, y_1^i, \dots, y_m^i)^i$ with $l_1, \dots, l_n > i$



Tiering and Safe recursion: same idea

It is forbidden to iterate a function which is itself defined by recursion.

More formally, in a recursive definition

$$f(s(x), y) = h(x, y, f(x, y))$$

the step function h is not allowed to recurse on the result of a previous function call, but may, however, recurse on other parameters.



Exploiting predicative recursion

Tiering has been used to characterize:

- ▶ **Polynomial Time** (Leivant)
- ▶ **Polynomial Space** (Leivant and Marion, Oitavem)
- ▶ **Alternating Logarithmic Time** (Leivant and Marion)



Higher-order functions

- ▶ A (programming) language has higher-order (functions) when functions can be both input and output of other functions.
- ▶ In presence of higher-order functions, we have exponential growth even without “recursion on recursive values” (which is what is forbidden by safe/tiered recursion).
- ▶ Consider the following higher-order function:

$$g(\epsilon) = s_0$$

$$g(s_0(x)) = g(x) \circ g(x)$$

$$g(s_1(x)) = g(x) \circ g(x)$$

$$\begin{aligned}g(b_k \cdots b_3 b_2 b_1) &= g(b_k \cdots b_3 b_2) \circ g(b_k \cdots b_3 b_2) \\ &= g(b_k \cdots b_3) \circ g(b_k \cdots b_3) \circ g(b_k \cdots b_3 b_2) \\ &= \dots \\ &= g(\epsilon) \circ \dots \circ g(\epsilon) \quad 2^k \text{ times}\end{aligned}$$



Exponential growth with higher-order

- ▶ We have defined

$$\begin{aligned}g(\epsilon) &= s_0 \\g(s_0(x)) = g(s_1(x)) &= g(x) \circ g(x)\end{aligned}$$

- ▶ $g(x) = s_0 \circ \dots \circ s_0$, $2^{|x|}$ times
- ▶ As numbers: $h(n)(y) = 2^{|x|} \cdot y$.
- ▶ Here there is no recursion on results of recursive calls. . .
- ▶ The problem seems to be in the **reuse** of an argument
- ▶ Here the step function is $h(z) = z \circ z$
- ▶ Impose some kind of **linearity** constraint.



Preliminaries: λ -calculus

- ▶ The language:

$$M, N ::= x \mid \lambda x.M \mid (MN)$$

- ▶ Notation:

- ▶ $\lambda x_1 x_2.M$ is $\lambda x_1.(\lambda x_2.M)$

- ▶ MNP is $((MN)P)$

- ▶ $M[N/x]$: the substitution of N for the free occurrences of x in M

- ▶ Beta contraction: $(\lambda x.M)N \rightarrow_{\beta} M[N/x]$

- ▶ Reduction (\rightarrow) is context, reflexive and transitive closure of beta contraction



Types for λ -terms

- ▶ The language of types:

$$T, S ::= o \mid T \rightarrow S$$

- ▶ Typing rules

$$x : T \vdash x : T \quad (Ax)$$

$$\frac{\Gamma, x : S \vdash M : T}{\Gamma \vdash \lambda x. M : S \rightarrow T} \quad (\mathcal{I} \rightarrow) \quad \frac{\Gamma \vdash M : S \rightarrow T \quad \Gamma \vdash N : S}{\Gamma \vdash MN : T} \quad (\mathcal{E} \rightarrow)$$



Fundamental properties

- ▶ This typed calculus is a very well behaved system.
- ▶ “**subject reduction**” (i.e., preservation of types under reduction): $\Gamma \vdash M : T$ and $M \rightarrow^* N$, then $\Gamma \vdash N : T$;
- ▶ **Confluence**: $M \rightarrow^* N_1$ and $M \rightarrow^* N_2$, then there exists P such that $N_1 \rightarrow^* P$ and $N_2 \rightarrow^* P$;
- ▶ Hence we have **unicity of normal forms**;
- ▶ **Strong normalization**: Any term has a normal form, which is obtained under any reduction strategy.



Add a base type for natural numbers

- ▶ The language of types:

$$T, S ::= \mathbb{N} \mid T \rightarrow S$$

- ▶ Terms: add new constants. E.g.,
 $0, s, cond$
- ▶ Typing rules: add type axioms for the new constants. E.g.,

$$\Gamma \vdash 0 : \mathbb{N} \quad \Gamma \vdash s : \mathbb{N} \rightarrow \mathbb{N}$$

$$\Gamma \vdash cond : \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}$$

- ▶ Reduction: add contraction rules for the new constants. E.g.,

$$cond\ 0\ M\ P \rightarrow_{\delta} M$$

$$cond\ (sN)\ M\ P \rightarrow_{\delta} P$$



A higher-order version of Cobham: PV^ω

- ▶ Cook & Urquhart 1993
- ▶ Typed λ -calculus over base type \mathbb{N} ;
- ▶ Constants on \mathbb{N} :
 - ▶ Zero: $0 : \mathbb{N}$;
 - ▶ successors $s_0, s_1 : \mathbb{N} \rightarrow \mathbb{N}$;
 - ▶ division by 2 $p : \mathbb{N} \rightarrow \mathbb{N}$, $p(n) = \lfloor n/2 \rfloor$;
 - ▶ smash $\#(x)(y) = 2^{|x| \cdot |y|}$;
 - ▶ pad (shift left): $pad(x)(y) = x \cdot 2^{|y|}$;
 - ▶ chop (shift right): $chop(x)(y) = \lfloor x/2^{|y|} \rfloor$;
 - ▶ conditional: $cond(x)(y)(z) = y$ if $x = 0$; otherwise $= z$.
- ▶ Bounded recursion: for $z, x : \mathbb{N}$, $h : \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}$, $k : \mathbb{N} \rightarrow \mathbb{N}$
 $f(x) = rec(z, h, k, x)$ is the function defined as

$$f(0) = \min(k(0), z)$$

$$f(x) = \min(k(x), h(x, f(p(x))))$$



- ▶ Prove by induction that for any $f(x_1, \dots, x_n)$ in Cobham there is a term $M_f : \mathbb{N}^n \rightarrow \mathbb{N}$ computing f .
- ▶ Being a typed lambda-calculus, it allows for direct definitions of higher-order functions.
- ▶ Example: $\exists : (\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N} \rightarrow \mathbb{N}$
 $\exists(f)(x)$ is the least $i \leq x$ s.t. $f(i) = 0$, if it exists, otherwise is $f(x)$.
 $\exists = \lambda f. \lambda x. \text{rec}(f(0), \lambda u. \lambda v. \text{cond}(v, 0, f(|x|)))$

Theorem

If $M : \mathbb{N}^n \rightarrow \mathbb{N}$ in PV^ω , then the function computed by M is computable in polytime.

- ▶ Same critique as for Cobham: can we do the same without initial polynomial functions **and** without explicit counting during recursion?



Typed Lambda-Calculi: Higher-Order Recursion

- ▶ Higher-order generalizations of Leivant's ramified recurrence captures elementary time computable functions (Leivant, Bellantoni Niggel Schwichtenberg, Dal Lago Martini Roversi)
- ▶ Polynomial time can be retrieved by constraining higher-order variables to be used in a linear way (Hofmann).
- ▶ Non-size increasing polytime computation is a calculus for polynomial time functions which uses a stricter notion of linearity, but without any ramification condition (Hofmann).
- ▶ Characterizations of major complexity classes can be obtained using syntactical constraints on lambda-calculi with higher-type recursion (Leivant).



Uniform approach, tailoring Gödel's T

- ▶ Gödel's **System T** is a well known typed λ -calculus with \mathbb{N} as base type and explicit recursion.
- ▶ Introduced for foundational purposes: to prove the consistency of Peano Arithmetic (the **Dialectica interpretation**, 1959).
- ▶ The terms in T with type $\mathbb{N} \rightarrow \mathbb{N}$ have huge computational power.

Theorem

$M : \mathbb{N} \rightarrow \mathbb{N}$ in T iff M computes a function provably total in Peano Arithmetic.

- ▶ We will see **simple** syntactic restrictions on T giving rise to interesting computational classes (Dal Lago, 2005).
- ▶ This summarizes many previous results into a single uniform setting.



Base types: free algebras

- ▶ A **free algebra** \mathbb{A} : constants (constructors) with their arity (given as a function $\mathcal{R}_{\mathbb{A}}$). Examples:
 - ▶ Unary naturals: $\mathbb{U} = \{0, s\}$; $\mathcal{R}_{\mathbb{U}}(0) = 0$ and $\mathcal{R}_{\mathbb{U}}(s) = 1$;
 - ▶ Binary naturals: $\mathbb{B} = \{\epsilon, s_0, s_1\}$; $\mathcal{R}_{\mathbb{B}}(\epsilon) = 0$ and $\mathcal{R}_{\mathbb{B}}(s_i) = 1$;
 - ▶ Binary trees: $\mathbb{C} = \{\epsilon, c\}$; $\mathcal{R}_{\mathbb{C}}(\epsilon) = 0$ and $\mathcal{R}_{\mathbb{C}}(c) = 2$;
- ▶ \mathbb{U} and \mathbb{B} are examples of **word algebras**.
- ▶ Fix a finite family \mathcal{A} of free algebras $\{\mathbb{A}_1, \dots, \mathbb{A}_n\}$, including \mathbb{U}, \mathbb{B} and \mathbb{C} .



Terms and reduction

- ▶ Terms over \mathcal{A}

$$M ::= x \mid c \mid MM \mid \lambda x.M \mid M \{M, \dots, M\} \mid M \langle\langle M, \dots, M \rangle\rangle$$

c ranges over the constants of \mathcal{A} ; $\{\dots\}$ is conditional; $\langle\langle \dots \rangle\rangle$ is recursion (after Matthes and Joachimsky, 2003).

- ▶ Reduction rules:

$$(\lambda x.M)V \rightarrow M\{V/x\}$$

$$c_i(t_1, \dots, t_{\mathcal{R}(c_i)})\{M_{c_1}, \dots, M_{c_k}\} \rightarrow M_{c_i} t_1 \cdots t_{\mathcal{R}(c_i)}$$

$$\begin{aligned} c_i(t_1, \dots, t_{\mathcal{R}(c_i)})\langle\langle M_{c_1}, \dots, M_{c_k} \rangle\rangle &\rightarrow M_{c_i} t_1 \cdots t_{\mathcal{R}(c_i)} \\ &\quad (t_1 \langle\langle M_{c_1}, \dots, M_{c_k} \rangle\rangle) \\ &\quad \dots \\ &\quad (t_{\mathcal{R}(c_i)} \langle\langle M_{c_1}, \dots, M_{c_k} \rangle\rangle) \end{aligned}$$

- ▶ Reduction is **not allowed**:
under abstractions, or inside $\{\}$ and $\langle\langle \rangle\rangle$.



The simple case of \mathbb{B}

- ▶ Conditional and recursion for the binary naturals:

$$\mathbb{B} = \{\epsilon, s_0, s_1\}; \mathcal{R}_{\mathbb{B}}(\epsilon) = 0 \text{ and } \mathcal{R}_{\mathbb{B}}(s_i) = 1$$

- ▶ Conditional:

$$\epsilon \{M_{\epsilon}, M_0, M_1\} \rightarrow M_{\epsilon}$$

$$s_0 t \{M_{\epsilon}, M_0, M_1\} \rightarrow M_0 t$$

$$s_1 t \{M_{\epsilon}, M_0, M_1\} \rightarrow M_1 t$$

- ▶ Recursion:

$$\epsilon \langle\langle M_{\epsilon}, M_0, M_1 \rangle\rangle \rightarrow M_{\epsilon}$$

$$s_0 t \langle\langle M_{\epsilon}, M_0, M_1 \rangle\rangle \rightarrow M_0 t (t \langle\langle M_{\epsilon}, M_0, M_1 \rangle\rangle)$$

$$s_1 t \langle\langle M_{\epsilon}, M_0, M_1 \rangle\rangle \rightarrow M_1 t (t \langle\langle M_{\epsilon}, M_0, M_1 \rangle\rangle)$$



Types

$$A ::= \mathbb{A}^n \mid A \multimap A$$

where n ranges over \mathbb{N} and \mathbb{A} ranges over \mathcal{A} . Indexing base types is needed to define tiering conditions.

$$\frac{}{x : A \vdash x : A} A \quad \frac{\Gamma \vdash M : B}{\Gamma, x : A \vdash M : B} W \quad \frac{\Gamma, x : A, y : A \vdash M : B}{\Gamma, z : A \vdash M\{z/x, z/y\} : B} C$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x. M : A \multimap B} I_{\multimap} \quad \frac{\Gamma \vdash M : A \multimap B \quad \Delta \vdash N : A}{\Gamma, \Delta \vdash MN : B} E_{\multimap}$$

$$\frac{n \in \mathbb{N} \quad c \in \mathcal{C}_{\mathbb{A}}}{\vdash c : \mathbb{A}^n \multimap \mathbb{A}^n} I_c \quad \frac{\Gamma_i \vdash M_{c_i^{\mathbb{A}}} : \mathbb{A}^m \multimap C \quad \Delta \vdash L : \mathbb{A}^m}{\Gamma_1, \dots, \Gamma_n, \Delta \vdash L \{M_{c_1} \cdots M_{c_k}\} : C} E_C^{\multimap}$$

$$\frac{\Gamma_i \vdash M_{c_i^{\mathbb{A}}} : \mathbb{A}^m \multimap C \quad \mathcal{R}_{\mathbb{A}}(c_i^{\mathbb{A}}) \quad \Delta \vdash L : \mathbb{A}^m}{\Gamma_1, \dots, \Gamma_n, \Delta \vdash L \langle\langle M_{c_1} \cdots M_{c_k} \rangle\rangle : C} E_{\multimap}^R$$



Expressive power

- ▶ Without restriction it is equivalent to Gödel's T (over free algebras)
- ▶ Indeed, if we take the only algebra \mathbb{U} of unary naturals, this is Gödel's T
- ▶ Restrictions. Two dimensions:
 - ▶ Tiering/stratification/ramification on the recursion rule, to ensure low computational power at first-order;
 - ▶ Linearity (i.e., contraction rule), to control the higher-order features.



Tiering constraints

In the rule

$$\frac{\Gamma_j \vdash M_{c_i^{\mathbb{A}}} : \mathbb{A}^m \quad \mathcal{R}_{\mathbb{A}}(c_i^{\mathbb{A}}) \quad C \quad \Delta \vdash L : \mathbb{A}^m}{\Gamma_1, \dots, \Gamma_n, \Delta \vdash L \langle\langle M_{c_1} \cdots M_{c_k} \rangle\rangle : C} E_{\rightarrow}^R$$

add the constraint

$$m > V(C)$$

where $V(C)$ is the maximum tier of a base type in C .



Linearity constraints

- ▶ The contraction rule

$$\frac{\Gamma, x : A, y : A \vdash M : B}{\Gamma, z : A \vdash M\{z/x, z/y\} : B} C$$

may be applied **only** to types in a class $\mathbf{D} \subseteq \mathcal{T}_{\mathcal{A}}$.

- ▶ In the recursion rule

$$\frac{\Gamma_i \vdash M_{c_i^A} : \mathbb{A}^m \quad \mathcal{R}_{\mathbb{A}}(c_i^A) \quad C \quad \mathcal{R}_{\mathbb{A}}(c_i^A) \quad C \quad \Delta \vdash L : \mathbb{A}^m}{\Gamma_1, \dots, \Gamma_n, \Delta \vdash L \langle\langle M_{c_1} \dots M_{c_k} \rangle\rangle : C} E_{\rightarrow}^R$$

$\text{cod}(\Gamma_i) \subseteq \mathbf{D}$ for every $i \in \{1, \dots, n\}$.



Several possible systems

- ▶ The unrestricted system: $\mathbf{H}(\mathcal{I}_{\mathcal{A}})$
- ▶ The system with contraction limited to \mathbf{D} : $\mathbf{H}(\mathbf{D})$
- ▶ The tiered (**ramified**) system: add \mathbf{R} to the name of the system; e.g., \mathbf{RH} , $\mathbf{RH}(\mathbf{D})$.
- ▶ We investigate the following \mathbf{D} 's:
 - ▶ The purely linear system: $\mathbf{D} = \emptyset$;
 - ▶ Contraction only on word algebras:
 $\mathbf{D} = \mathbf{W} = \{\mathbb{A}^n \mid \mathbb{A} \in \mathcal{A} \text{ is a word algebra}\}$;
 - ▶ Contraction only on base types (algebras):
 $\mathbf{D} = \mathbf{A} = \{\mathbb{A}^n \mid \mathbb{A} \in \mathcal{A}\}$



And their expressive power

	H(\emptyset)	H(W)	H(A)
no ramification	Prim. Rec.	Prim. Rec.	Prim. Rec.
ramification	PolyTime	PolyTime	ElementaryTime
	RH(\emptyset)	RH(W)	RH(A)

- ▶ Any term of one of the systems can be normalized within the associated time bound.
- ▶ For any function f of one of the complexity classes, there exists a term M_f computing f which, in the associated system, has type $\mathbb{A}^n \rightarrow \mathbb{A}$.
- ▶ Recall that in $\mathbf{H}(\mathcal{T}_{\mathcal{A}})$ we characterize all functions provably total in Peano Arithmetic.



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

$$A \vdash A \text{ (Ax)}$$

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

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

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

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

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

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

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

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

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



The Curry-Howard correspondence: Annotated proofs

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

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

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

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

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

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

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

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

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

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



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: $\lambda x^t.x : \forall t.t \rightarrow t$;
- ▶ Natural numbers: $\mathbb{N} = \forall t.(t \rightarrow t) \rightarrow (t \rightarrow t)$;
- ▶ The number 3: $\underline{3} = \lambda f^{t \rightarrow t}.\lambda x^t.f(f(fx)) : \mathbb{N}$
These are the **Church numerals**. In general:
 $\underline{n} = \lambda f^{t \rightarrow t}.\lambda x^t.f^n x : \mathbb{N}$
- ▶ Binary words: $\mathbb{B} = \forall t.(t \rightarrow t) \rightarrow (t \rightarrow t) \rightarrow (t \rightarrow t)$;
- ▶ The binary word 01 (that is: $s_0 s_1 \epsilon$): $\lambda s_0^{t \rightarrow t}.\lambda s_1^{t \rightarrow t}.\lambda e^t.s_0(s_1 e)$;
- ▶ 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 $F : T \rightarrow T$.
 - ▶ For any $a : T$ we have $\underline{n} F a \rightarrow F(F \cdots (Fa) \cdots)$, with n occurrences of F .
 - ▶ $iter_T = \lambda n. \lambda f. \lambda x. n f x : \mathbb{N} \rightarrow (T \rightarrow T) \rightarrow T \rightarrow T$.
 - ▶ A doubling function:
 $double = \lambda n. \underline{2n} : \mathbb{N} \rightarrow \mathbb{N}$;
 - ▶ An exponential function:
 $exp = \lambda n. iter_{\mathbb{N}} n double \underline{1} : \mathbb{N} \rightarrow \mathbb{N}$



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 $\mathbb{N}_0 = (\alpha \rightarrow \alpha) \rightarrow (\alpha \rightarrow \alpha)$ 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

$$A \vdash A \text{ (Ax)}$$

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

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

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

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

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

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

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



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. . .



Adding Exponentials: I(ME)LL

$$A \vdash A \text{ (Ax)}$$

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

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

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

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

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

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

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

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

$$\frac{!A_1, \dots, !A_n \vdash B}{!A_1, \dots, !A_n \vdash !B} \text{ (!,r)}$$

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

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



A variant

$$A \vdash A \text{ (Ax)}$$

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

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

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

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

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

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

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

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

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

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

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

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



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	$!A \multimap A \otimes \dots \otimes A$	

...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 + (\S)
SLL	YES	NO	NO	NO	YES	derivable

where

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

$$\frac{\Gamma, A \vdash B}{\Gamma, !A \vdash B} \text{ (ϵ)} \qquad \frac{\Gamma, !!A \vdash B}{\Gamma, !A \vdash B} \text{ (δ)}$$

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



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 Grzegorzcyk 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

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

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

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

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

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

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

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

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

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



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 $\Pi_M \rightarrow \Pi'$ in normal form. Then Π' corresponds to a proof of $\Gamma \vdash M' : A$, with $M \rightarrow M'$ 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 $2^{\cdot^{|\Pi|}} \}^{d_\Pi}$ 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

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

- ▶ 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, \S , with rule

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



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 $\mathbb{B} \multimap \mathbb{S}^k \mathbb{B}$) defining f .



