Implicit Computational Complexity

Simone Martini

Dipartimento di Scienze dell'Informazione
Università di Bologna
Italy

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

Outline: first part

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);
  - $\textsc{Resource}[f(n)] =$ $\{P \mid \text{there exists TM } M \text{ deciding } P \text{ and working in bound } f\}$;
- Some relevant classes
  - $\textsc{LogSpace} = \textsc{Space}[\log n]$;
  - $\textsc{LinTime} = \textsc{Time}[n]$;
  - $\textsc{PTime} = \cup_{i \in N} \textsc{Time}[n^i]$;
  - $\textsc{PSpace} = \cup_{i \in N} \textsc{Space}[n^i]$;
  - $\textsc{ExpTime} = \textsc{Time}[2^n]$;

Invariance

- Classes are invariant w.r.t. linear factors: $\textsc{Resource}[f(n)] = \textsc{Resource}[af(n) + b]$;
- 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, $\textsc{PTime}$ and $\textsc{PSpace}$ 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 lenght of the input would be esponentially longer than the lenght in any other base. Therefore giving esponentially more resource to the computation. (Remember: the bound is a function of $|u|$).

Functional classes

- $\textsc{FPtime} =$
  $\{f : \mathbb{N} \to \mathbb{N} \mid$
  there exists TM M computing f in polynomial bound$\}$;
- $\textsc{FLogSpace} = \ldots;$
- $\ldots$

Machine-free definitions of functions: Gödel-Kleene

Class of n-ary functions defined by closure.

- Base functions:
  - Constant zero: $Z : \mathbb{N} \to \mathbb{N}$, $Z(y) = 0$;
  - Successor: $S : \mathbb{N} \to \mathbb{N}$, $S(y) = y + 1$;
  - Projections: for any $k \in \mathbb{N}$ and $i \leq k$, $\pi_i^k : \mathbb{N}^k \to \mathbb{N}$, $\pi_i^k(y_1, \dots, y_k) = y_i$.
- 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

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

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

$$\begin{aligned} f(\overline{y}) \quad = \quad & \text{the } \textit{least } z \text{ such that (i) } g(z, \overline{y}) = 0 \text{ and} \\ & \text{(ii) } g(x, \overline{y}) \text{ is defined for all } x \leq z \end{aligned}$$

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

- 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 $\mathrm{FPT}\textsc{ime}$.
- We first study some known subclasses of the primitive recursive functions.

The spine of primitive recursion

$$
\begin{aligned}
f_0(x,y) &= x+1; \\
f_1(x,y) &= x+y; \\
f_2(x,y) &= xy; \\
f_{n+1}(x,0) &= 1; \\
f_{n+1}(x,y+1) &= f_n(x, f_{n+1}(x,y))
\end{aligned}
$$

$$
\begin{aligned}
f_3(x,y) &= x^y; \\
f_4(x,y) &= x^{x^{\cdot^{\cdot^{\cdot^{x}}}}}\Big\}{y \text{ times}}.
\end{aligned}
$$

Theorem

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

Grzegorczyk

- Recursion causes bigger growth than composition:
  - Define $f^k(x) = (f \circ \cdot \circ f)(x)$, 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 $\overline{x}$,

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

- 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.

Grzegorczyk 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 $h, g_1, \ldots, g_k$ have rank $\max\{\delta(h), \delta(g_1), \ldots, \delta(g_k)\}$;
  - 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 classs 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:

$$
\begin{array}{rcl}
f(0, \overline{y}) & = & g_0(\overline{y}) \\
f(1, \overline{y}) & = & g_1(\overline{y}) \\
f(s_0(x), \overline{y}) & = & h_0(x, \overline{y}, f(x, \overline{y})) \\
f(s_1(x), \overline{y}) & = & h_1(x, \overline{y}, f(x, \overline{y}))
\end{array}
$$

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):

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

As numbers ($*$ is multiplication):

$$\begin{aligned} d(0) = d(1) &= 1 \\ d(n) &= 4 * d(\lfloor x/2 \rfloor) \end{aligned}$$

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

Recursion on notation is too generous

Recall

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

And define

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

Now $e(x)$ has exponential lenght in $|x|$...

Still too much growth...

Bounded recursion on notation

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

$$
\begin{aligned}
f(0, \overline{y}) &= g_0(\overline{y}) \\
f(1, \overline{y}) &= g_1(\overline{y}) \\
f(s_0(x), \overline{y}) &= h_0(x, \overline{y}, f(x, \overline{y})) \\
f(s_1(x), \overline{y}) &= h_1(x, \overline{y}, f(x, \overline{y}))
\end{aligned}
$$

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

Cobham characterization of $\mathrm{FP}_{\mathrm{TIME}}$

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

Theorem (Cobham)

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

- Proof: $\mathcal{F}PTIME \subseteq \mathcal{COB}$: Code TMs as functions of the algebra. The iteration of the transition function is representable because *a priori* polynomially bounded. $\mathcal{COB} \subseteq \mathcal{F}PTIME$: 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

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

$$\begin{array}{rcl} f(0, \overline{y}) & = & g_0(\overline{y}) \\ f(1, \overline{y}) & = & g_1(\overline{y}) \\ f(s_0(x), \overline{y}) & = & h_0(x, \overline{y}, f(x, \overline{y})) \\ f(s_1(x), \overline{y}) & = & h_1(x, \overline{y}, f(x, \overline{y})) \end{array}$$

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

Logspace

Theorem (Lind;Clote & Takeuti)

$\mathcal{F}$LOGSPACE 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 ($h_i(x, \overline{y}) \leq 1$) is

$$\begin{aligned} f(0, \overline{y}) &= g_0(\overline{y}) \\ f(1, \overline{y}) &= g_1(\overline{y}) \\ f(s_0(x), \overline{y}) &= s_{h_0(x,\overline{y})}(f(x, \overline{y})) \\ f(s_1(x), \overline{y}) &= s_{h_1(x,\overline{y})}(f(x, \overline{y})) \end{aligned}$$

A critique on Cobham characterization

- Cobham's paper is the birth of computational complexity as a respected theory.
- It characterized PTIME 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 PTIME...

Safe Recursion: idea

- Unbounded recursion schema to control the growth of functions
- Function arguments are partioned into two separate classes.
- Function definitions are constrained to respect this partition.
- The arguments to a function $f : \mathbb{N}^n \to \mathbb{N}$ 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 $g, h_1, \ldots, h_n, k_1, \ldots, k_m$ if

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

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

$$\begin{aligned}
f(0,\overline{x};\overline{y}) &= g_0(\overline{x};\overline{y}) \\
f(1,\overline{x};\overline{y}) &= g_1(\overline{x};\overline{y}) \\
f(s_0(x),\overline{x};\overline{y}) &= h_0(x,\overline{x};\overline{y}, f(x,\overline{x};\overline{y})) \\
f(s_1(x),\overline{x};\overline{y}) &= h_1(x,\overline{x};\overline{y}, f(x,\overline{x};\overline{y}))
\end{aligned}$$

Understanding safe composition and recursion

- The key clause:

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

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

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

    and will not be copied back into a normal position.

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

Projections

- We have projections from both normal and safe zones

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

- 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:

$$\begin{aligned} d(0;) = d(1;) &= 1 \\ d(s_0(x);) = d(s_1(x);) &= s_0(;s_0(;d(x;))) \\ & \quad \text{where formally the step function } h \text{ is} \\ h(x;z) &= \pi^2_2(x;s_0(;s_0(;\pi^2_2(x;z)))) \end{aligned}$$

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:

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

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 $|f(\overline{x};\overline{y})| \leq q_f(|\overline{x}|) + \max(y_1, \dots, y_n)$
  - 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 $f(\overline{y})$ there exists a function $f' \in \mathcal{B}$ and a polynomial p_f such that $f'(w;\overline{y}) = f(\overline{y})$, for all $\overline{y}$ and all $w \geq p_f(|\overline{y}|)$
  - Now construct a function b in $\mathcal{B}$ such that $b(\overline{x};) \geq p_f(|\overline{x}|)$
  - Set $f(\overline{x};) = f'(b(\overline{x};);\overline{x})$.

Variations: Safe *Affine* Composition

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

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

- 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 $g, h_1, \ldots, h_n, k_1, \ldots, k_m$ if

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

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

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

$$
\begin{array}{rcl}
f(0, \overline{x} : \overline{y}) & = & g_0(\overline{x} : \overline{y}) \\
f(1, \overline{x} : \overline{y}) & = & g_1(\overline{x} : \overline{y}) \\
f(s_0(x), \overline{x} : \overline{y}) & = & h_0(x, \overline{x} : f(x', \overline{x} : \overline{y})) \\
f(s_1(x), \overline{x} : \overline{y}) & = & h_1(x, \overline{x} : f(x'', \overline{x} : \overline{y})) \text{ with } x', x'' \leq x
\end{array}
$$

Theorem (Mairson and Neergaard, 2003)

The set of logaritmic 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:
  $\mathbb{N}^0, \mathbb{N}^1, \mathbb{N}^2, \ldots$
- Functions have a type of the form
  $f : \mathbb{N}^i \times \cdots \times \mathbb{N}^j \to \mathbb{N}^k$
- 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:

$$
\begin{aligned}
f(0, y)^i &= g_0(y^k)^i \\
f(1, y)^i &= g_1(y^k)^i \\
f(s_0(x)^l, y)^i &= h_0(x^l, y^k, f(x, y)^i)^i \\
f(s_1(x)^l, y)^i &= h_1(x^l, y^k, f(x, y)^i)^i \text{ with } l > i
\end{aligned}
$$

- 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
  $p(s_b(x)) = x$
- Concatenation:

$$\begin{array}{rcl} \oplus(\epsilon, y) & = & y \\ \oplus(s_b(x), y) & = & s_b(\oplus(x,y)) \end{array}$$

Imposing stratification:
$\oplus(s_b(x)^l, y^j)^i = s_b(\oplus(x^l, y^j)^i)$ with $l > i$
Take $l = 1$, $i = 0$ (and j whatever, say 0):
$\oplus : \mathbb{N}^1 \times \mathbb{N}^0 \to \mathbb{N}^0$

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:
  $+(s(x)^1, y^0)^1 = s(+(x^1, y^0)^1)$
  $+ : \mathbb{N}^1 \times \mathbb{N}^0 \to \mathbb{N}^0$

- 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: $* : \mathbb{N}^1 \times \mathbb{N}^1 \to \mathbb{N}^0$

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 $P2(n) = 2^n$:

$$\begin{aligned} P2(0) &= 1 \\ P2(s(x)) &= +(P2(x), P2(x)) \end{aligned}$$

Recall that $+ : \mathbb{N}^1 \times \mathbb{N}^0 \to \mathbb{N}^0$
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 $f(x_1^{l_1}, \ldots, x_n^{l_n}, y_1^i, \ldots y_m^i)^i$ where $l_1, \ldots, l_n > i$ we get $f(x_1, \ldots, x_n; y_1, \ldots, y_m)$

  - From a safe definition $f(x_1, \ldots, x_n; y_1, \ldots, y_m)$ for any tier i, there is a tiered definition of f in which $f(x_1^{l_1}, \ldots, x_n^{l_n}, y_1^i, \ldots y_m^i)^i$ with $l_1, \ldots, l_n > i$

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 forbidded by safe/tiered recursion).
- Consider the following higher-order function:

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

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

Exponential growth with higher-order

- We have defined

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

- $g(x) = s_0 \circ \cdots \circ s_0$, $2^{|x|}$ 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

$$x : T \vdash x : T \ \scriptstyle{(Ax)}$$

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

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 $M \longrightarrow^* N$, then $\Gamma \vdash N : T$;
- Confluence: $M \longrightarrow^* N_1$ and $M \longrightarrow^* N_2$, then there exists P such that $N_1 \longrightarrow^* P$ and $N_2 \longrightarrow^* P$;
- 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:

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

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

$$\Gamma \vdash 0 : \mathbb{N} \qquad \Gamma \vdash s : \mathbb{N} \to \mathbb{N}$$

$$\Gamma \vdash \mathit{cond} : \mathbb{N} \to \mathbb{N} \to \mathbb{N} \to \mathbb{N}$$

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

$$\mathit{cond}\ 0\ M\ P \to_\delta M$$

$$\mathit{cond}\ (sN)\ M\ P \to_\delta P$$

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 $s_0, s_1 : \mathbb{N} \to \mathbb{N}$;
  - division by 2 $p : \mathbb{N} \to \mathbb{N}$, $p(n) = \lfloor n/2 \rfloor$;
  - 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 $z, x : \mathbb{N}$, $h : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$, $k : \mathbb{N} \to \mathbb{N}$ $f(x) = \mathrm{rec}(z, h, k, x)$ is the function defined as

$$
\begin{aligned}
f(0) &= \min(k(0), z) \\
f(x) &= \min(k(x), h(x, f(p(x))))
\end{aligned}
$$

PV^{ω}

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

Theorem

If $M : \mathbb{N}^n \to \mathbb{N}$ 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 Niggl Schwichtenberg, Dal Lago Martini Roversi)
- Polynomial time can be retrived 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).

Other higher-order systems

We will see the non size increasing calculus on Friday

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 $\mathbb{N} \to \mathbb{N}$ have huge computational power.

Theorem

$M : \mathbb{N} \to \mathbb{N}$ 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 $\mathscr{A}$ of free algebras $\{\mathbb{A}_1, \ldots, \mathbb{A}_n\}$, including $\mathbb{U}, \mathbb{B}$ and $\mathbb{C}$.

Terms and reduction

- Terms over $\mathscr{A}$

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

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

- Reduction rules:

$$\begin{array}{rcl}
(\lambda x.M)V & \to & M\{V/x\} \\
c_i(t_1,\ldots,t_{\mathcal{R}(c_i)})\{\!\{M_{c_1},\ldots,M_{c_k}\}\!\} & \to & M_{c_i}\ t_1\cdots t_{\mathcal{R}(c_i)} \\
c_i(t_1,\ldots,t_{\mathcal{R}(c_i)})\langle\!\langle M_{c_1},\ldots,M_{c_k}\rangle\!\rangle & \to & M_{c_i}\ t_1\cdots t_{\mathcal{R}(c_i)} \\
 & & \quad (t_1\ \langle\!\langle M_{c_1},\ldots,M_{c_k}\rangle\!\rangle) \\
 & & \quad \ldots \\
 & & \quad (t_{\mathcal{R}(c_i)}\ \langle\!\langle M_{c_1},\ldots,M_{c_k}\rangle\!\rangle)
\end{array}$$

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

$$\begin{array}{rcl} \epsilon \, \{\!\!\{ M_\epsilon, M_0, M_1 \}\!\!\} & \rightarrow & M_\epsilon \\ s_0 t \, \{\!\!\{ M_\epsilon, M_0, M_1 \}\!\!\} & \rightarrow & M_0 t \\ s_1 t \, \{\!\!\{ M_\epsilon, M_0, M_1 \}\!\!\} & \rightarrow & M_1 t \end{array}$$

- Recursion:

$$\begin{array}{rcl} \epsilon \, \langle\!\langle M_\epsilon, M_0, M_1 \rangle\!\rangle & \rightarrow & M_\epsilon \\ s_0 t \, \langle\!\langle M_\epsilon, M_0, M_1 \rangle\!\rangle & \rightarrow & M_0 t \, (t \, \langle\!\langle M_\epsilon, M_0, M_1 \rangle\!\rangle) \\ s_1 t \, \langle\!\langle M_\epsilon, M_0, M_1 \rangle\!\rangle & \rightarrow & M_1 t \, (t \, \langle\!\langle M_\epsilon, M_0, M_1 \rangle\!\rangle) \end{array}$$

Types

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

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

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

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

$$\frac{n \in \mathbb{N} \quad c \in \mathcal{C}_{\mathbb{A}}}{\vdash c : \mathbb{A}^n \overset{\mathcal{R}_{\mathbb{A}}(c)}{\multimap} \mathbb{A}^n}\ I_c \qquad \frac{\Gamma_i \vdash M_{c_i^{\mathbb{A}}} : \mathbb{A}^m \overset{\mathcal{R}_{\mathbb{A}}(c_i^{\mathbb{A}})}{\multimap} C \quad \Delta \vdash L : \mathbb{A}^m}{\Gamma_1, \ldots, \Gamma_n, \Delta \vdash L\,\{M_{c_1} \cdots M_{c_k}\} : C}\ E^C_{\multimap}$$

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

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

$$\frac{\Gamma_i \vdash M_{c_i^{\mathbb{A}}} : \mathbb{A}^m \overset{\mathcal{R}_{\mathbb{A}}(c_i^{\mathbb{A}})}{\multimap} C \overset{\mathcal{R}_{\mathbb{A}}(c_i^{\mathbb{A}})}{\multimap} C \quad \Delta \vdash L : \mathbb{A}^m}{\Gamma_1, \ldots, \Gamma_n, \Delta \vdash L \left\langle\!\left\langle M_{c_1} \cdots M_{c_k} \right\rangle\!\right\rangle : C} \; E^R_{\multimap}$$

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 $\mathbf{D} \subseteq \mathscr{T}_{\mathscr{A}}$.
- In the recursion rule

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

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

Several possible systems

- The unrestricted system: $\mathbf{H}(\mathscr{T}_{\mathscr{A}})$
- 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: $\mathbf{D} = \mathbf{W} = \{\mathbb{A}^n \mid \mathbb{A} \in \mathscr{A} \text{ is a word algebra}\}$;
  - Contraction only on base types (algebras): $\mathbf{D} = \mathbf{A} = \{\mathbb{A}^n \mid \mathbb{A} \in \mathscr{A}\}$

And their expressive power

	$\mathbf{H}(\emptyset)$	$\mathbf{H}(\mathbf{W})$	$\mathbf{H}(\mathbf{A})$
no ramification	Prim. Rec.	Prim. Rec.	Prim. Rec.
ramification	PolyTime	PolyTime	ElementaryTime
	$\mathbf{RH}(\emptyset)$	$\mathbf{RH}(\mathbf{W})$	$\mathbf{RH}(\mathbf{A})$

- 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 $\mathbf{H}(\mathscr{T}_{\mathscr{A}})$ we characterize all functions provably total in Peano Arithmetic.