Ramification

Jean-Yves Marion

Ecole nationale supérieure des Mines de Nancy
Loria-INPL

February, 6th 2006

Outline

Primitive recursion

A first order structure for unary numbers

- $\mathbf{Nat} = \langle 0, \mathbf{suc} \rangle$

is a set of data objects defined by

- O is a number of **Nat**
- if n is a number, then $\mathbf{suc}(n)$ is a number of **Nat**

Semantics

$$[\![\mathbf{Nat}]\!] = \mathbb{N}$$

Primitive recursion over **Nat**

$$f(0, \overline{x}) = g(\overline{x})$$
$$f(\mathbf{suc}(n), \overline{x}) = h(n, \overline{x}, f(n, \overline{x}))$$

Primitive recursion on Words

A first order structure for binary words

$$\textbf{Word} = \langle \epsilon, \mathbf{0}, \mathbf{1} \rangle$$

is a set of data objects defined by

- ϵ is a word of **Word**
- if u is a word, then $\mathbf{0}(u)$ and $\mathbf{1}(u)$ are words of **Word**

Semantics

$$[\![\textbf{Word}]\!] = \{0, 1\}^*$$

Primitive recursive functions on Words

The class of primitive recursive functions over **Word** contains

- Constructors of **Word**

$$x \mapsto \epsilon \qquad x \mapsto \mathbf{0}(x) \qquad x \mapsto \mathbf{1}(x)$$

- Projections : $\pi_i(x_1, \ldots, x_n) = x_i$

is closed under

- Composition

$$f(\overline{x}) = h(g_1(\overline{x}), \ldots, g_k(\overline{x}))$$

  where $\overline{x} = x_1, \ldots, x_n$
- and Primitive recursion over **Word**

$$\begin{aligned} f(\epsilon, \overline{x}) &= g(\overline{x}) \\ f(\mathbf{0}(w), \overline{x}) &= h_0(w, \overline{x}, f(w, \overline{x})) \\ f(\mathbf{1}(w), \overline{x}) &= h_1(w, \overline{x}, f(w, \overline{x})) \qquad \overline{x} = x_1, \ldots, x_n \end{aligned}$$

  - recurrence parameter : w
  - recursive call : $f(w, \overline{x})$

Word concatenation or addition

Example

$$
\begin{aligned}
\mathrm{add}(\epsilon, x) &= x \\
\mathrm{add}(\mathbf{0}(w), x) &= \mathbf{0}(\mathrm{add}(w, x)) \\
\mathrm{add}(\mathbf{1}(w), x) &= \mathbf{1}(\mathrm{add}(w, x))
\end{aligned}
$$

$$[\![\mathrm{add}]\!] : (\{0,1\}^*)^2 \mapsto \{0,1\}^*$$

$$\mathrm{add}(\mathbf{1}(\mathbf{0}(\epsilon)), v) = \mathbf{1}(\mathbf{0}(v))$$

Primitive recursion on an arbitrary structure Σ

A first order structure $\Sigma = \langle a_1, \ldots, a_n, b_1, \ldots, b_m \rangle$
Primitive recursion over Σ

$$
\begin{aligned}
f(a_i, \overline{x}) &= g_i(\overline{x}) && i = 1, n \\
f(b_j(w_1, \ldots, w_n), \overline{x}) &= h_j(\overline{w}, \overline{x}, f(w_1, \overline{x}), \ldots \\
&\qquad \ldots, f(w_n, \overline{x})) && j = 1, m
\end{aligned}
$$

Theorem

The class of primitive recursive functions over Σ is exactly the set of primitive recursive functions over natural numbers.

Bounded recursion

- G_0 is the class containing zero, suc., projections and closed under composition and bounded recursion:

$$
\begin{aligned}
f(0,\overline{x}) &= g(\overline{x}) \\
f(\mathbf{suc}(t),\overline{x}) &= h(t,\overline{x},f(t,\overline{x})) \\
f(t,\overline{x}) &\leq k(t,\overline{x}) \qquad \text{for } k \text{ is in } G_0
\end{aligned}
$$

Grzegorczyk Hierarchy

$$E_0(x,y) = x + y \qquad E_1(x) = x^2 + 1$$
$$E_{n+2}(0) = 2 \qquad E_{n+2}(x+1) = E_{n+1}(E_{n+2}(x))$$

- G_{n+1} is the class containing zero, **suc**, projections, E_n and closed under composition and bounded recursion:

$$\begin{aligned} f(0,\overline{x}) &= g(\overline{x}) \\ f(\mathbf{suc}(t),\overline{x}) &= h(t,\overline{x},f(t,\overline{x})) \\ f(t,\overline{x}) &\leq k(t,\overline{x}) \qquad \text{for } k \text{ is in } G_{n+1} \end{aligned}$$

Theorem

The union $\cup_n G_n$ is the class of P.R. functions.

- G_3 is the class of elementary functions (Kalmar)

PTIME

Definition

PTIME is the set of functions which are computed in polynomial time with a Turing machine.

- PTIME computationally tractable problems, *Cook's thesis*
- All reasonable formalizations of the intuitive notion of tractable computability are equivalent within a polynomial bounded overhead
- Polynomial-time Turing machines computability capture all tractable functions.

Bounded recursion overs words

The class $\mathcal{L}$ contains

- Constructors of **Word**

$$x \mapsto \epsilon \qquad x \mapsto \mathbf{0}(x) \qquad x \mapsto \mathbf{1}(x)$$

- Projections : $\pi_i(x_1, \ldots, x_n) = x_i$
- The *smash* function $x \# y = 2^{|x| \cdot |y|}$ where $|x|$ is the length of x.

and is closed under

- composition
- and bounded recursion

$$\begin{aligned} f(\epsilon, \overline{x}) &= g(\overline{x}) \\ f(\mathbf{0}(t), \overline{x}) &= h(t, \overline{x}, f(t, \overline{x})) \\ f(\mathbf{1}(t), \overline{x}) &= h(t, \overline{x}, f(t, \overline{x})) \\ f(t, \overline{x}) &\leq k(t, \overline{x}) \qquad k \text{ is in } \mathcal{L} \end{aligned}$$

Cobham's Characterization of Ptime

$$
\begin{aligned}
f(\epsilon, \overline{x}) &= g(\overline{x}) \\
f(\mathbf{0}(t), \overline{x}) &= h(t, \overline{x}, f(t, \overline{x})) \\
f(\mathbf{1}(t), \overline{x}) &= h(t, \overline{x}, f(t, \overline{x})) \\
f(t, x) &\leq k(t, \overline{x}) \qquad\qquad k \text{ is in } \mathcal{L}
\end{aligned}
$$

Based on Ritchies's work:

Theorem (Cobham (65))

The class $\mathcal{L}$ is exactly the class of PTIME *of functions which are computable in Polynomial time.*

Bounded recursion as a complexity model

- A lof of characterizations of complexity classes follow the Cobham's idea. See the survey of Clote.
- Polynomial resource bound is inside the $\mathcal{L}$'s formalization
- Not intrinsic : Separate resources from algorithms
- Applications
  - Difficult to show that a program is PTIME
  - Difficult to extract complexity bounds

References

A. Cobham.
The intrinsic computational difficulty of functions.
In Y. Bar-Hillel, editor, *Proc of the Int. Conf. on Logic, Methodology, and Philosophy of Science*, pages 24–30. North-Holland, 1962.

R. Péter.
Recursive Functions,
1966.
Academic Press

R. Ritchie.
Classes of recursive functions based on Ackermann's function.
Pacific journal of mathematics, 15(3), 1965.

H.E. Rose.
Subrecursion.
Oxford university press, 1984.

P.R. Global functions

- Interpret $[\![\mathbf{Nat}]\!] = \{0, \ldots, n\}$
- See f is a primitive recursive schema
- Define $[\![f]\!]_n$ as the interpretation of f over $\{0, \ldots, n\}$ where

$$\mathbf{suc}(m) = \begin{cases} m+1 & \text{if } m < n \\ n & \text{if } n = m \end{cases}$$

A global function F is defined from a primitive recursive schema f

$$F(n, \overline{x}) = [\![f]\!]_n(\overline{x}) \qquad\qquad x_i \leq n$$

Theorem (Gurevich)

The set of global functions is exactly the set of LOGSPACE *functions*

P.R. Global functions and PTIME

- Sazonov and Gurevich characterize PTIME using the Herbrand-Gödel equations over finite structures.
- Jones characterizes PTIME using cons-free while language.

reference

Y. Gurevich.
Algebras of feasible functions.
In *FOCS*, pages 210–214, 1983.

N. Jones.
LOGSPACE and PTIME characterized by programming languages.
Theoretical Computer Science, 228:151–174, 1999.

N. Jones.
The expressive power of higher order types or, life without cons.
Journal of Functional Programming, 11(1):55–94, 2000.

V. Sazonov.
Polynomial computability and recursivity in finite domains.
Elektronische Informationsverarbeitung und Kybernetik, 7:319–323, 1980.

Domains with two colors

Two first order structures for binary words

$$\mathbf{Word} = \langle \epsilon, \mathbf{0}, \mathbf{1} \rangle \qquad \text{Normal}$$

$$\mathbf{Word} = \langle \epsilon, \mathbf{0}, \mathbf{1} \rangle \qquad \text{Safe}$$

Functions over domains with colors :

$$[\![f]\!] : [\![\mathbf{Word}]\!]^p \times [\![\mathbf{Word}]\!]^q \to [\![\mathbf{Word}]\!]$$

$$\overline{x}, \overline{y} \to f(\overline{x}; \overline{y})$$

Note the semicolon ; separates arguments

Safe Composition and Recursion

- safe composition

$$f(\overline{x};\overline{y}) = g(h_1(\overline{x};);h_2(\overline{x};\overline{y}))$$

- safe recursion

$$\begin{aligned} f(\epsilon,\overline{x};\overline{y}) &= g(\overline{x};\overline{y}) \\ f(\mathbf{0}(z),\overline{x};\overline{y}) &= h_0(z,\overline{x};f(z,\overline{x};\overline{y}),\overline{y}) \\ f(\mathbf{1}(z),\overline{x};\overline{y}) &= h_1(z,\overline{x};f(z,\overline{x};\overline{y}),\overline{y}) \end{aligned}$$

Recursive calls are safe !!

Data ramification

Data Ramification implies

$$\mathbf{Word} > \mathbf{Word}$$

because

$$f(x;) = g(;I(x;)) \qquad \text{Safe comp}$$
$$I(x;) = x \qquad \text{projection}$$

But the converse does not hold !!

Safe Recursive functions

The class $\mathcal{B}$ of *safe recursive functions* contains
Safe basic functions

- Constructors : $x \mapsto \epsilon$, $x \mapsto \mathbf{0}(; x)$, and $x \mapsto \mathbf{1}(; x)$
- Predecessor : $p(; \epsilon) = \epsilon$, $p(; \mathbf{i}(; x)) = x$
- Conditional : $C(; x, y, z) = \begin{cases} y & \text{if } x = \mathbf{0}(x') \\ z & \text{otherwise} \end{cases}$
- Projections : $\pi_i(x_1, \ldots, x_n; x_{n+1}, \ldots, x_{n+m}) = x_i$

and is closed

- safe composition
- safe recursion

Examples

Concatenation or addition :

$$\begin{aligned} \mathtt{add}(\epsilon; x) &= x \\ \mathtt{add}(\mathbf{0}(w); x) &= \mathbf{0}(; \mathtt{add}(w; x)) \\ \mathtt{add}(\mathbf{1}(w); x) &= \mathbf{1}(; \mathtt{add}(w; x)) \end{aligned}$$

Multiplication by iterating addition

$$\begin{aligned} \mathtt{mul}(\epsilon, y;) &= \epsilon \\ \mathtt{mul}(\mathbf{0}(v), y;) &= \mathtt{add}(y; \mathtt{mul}(v, y;)) \\ \mathtt{mul}(\mathbf{1}(v), y;) &= \mathtt{add}(y; \mathtt{mul}(v, y;)) \end{aligned}$$

Theorem (Bellantoni-Cook)

The set PTIME *of functions which are computable in polynomial time is exactly the class* $\mathcal{B}$ *of safe recursive functions.*

- *Simmons (88)* was the first to suggest this data-separation for primitive recursion.

Exponential is not safe !

Double length function is safe

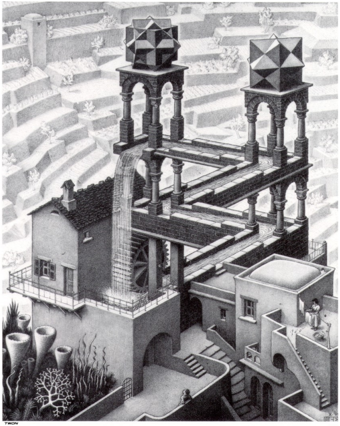
$$\begin{aligned}\mathtt{double}(\epsilon;) &= \epsilon\\ \mathtt{double}(\mathbf{0}(w);) &= \mathbf{1}(;\mathbf{1}(;\mathtt{double}(w;)))\\ \mathtt{double}(\mathbf{1}(w);) &= \mathbf{1}(;\mathbf{1}(;\mathtt{double}(w;)))\end{aligned}$$

Exponential by doubling is not safe

$$\begin{aligned}\mathtt{exp}(\epsilon;) &= \mathbf{1}(\epsilon)\\ \mathtt{exp}(\mathbf{0}(v);) &= \mathtt{double}(\mathtt{exp}(v;))\\ \mathtt{exp}(\mathbf{1}(v);) &= \mathtt{double}(\mathtt{exp}(v;))\end{aligned}$$

The recursive call $\mathtt{exp}(v;)$ should be safe. But `double` requires a normal argument.

A well-known Escher drawing to make a break



$$f : \mathbf{Word}(n+1) \rightarrow \mathbf{Word}(n)$$

Analysis of the energy of arguments in recursive definitions

Domains is stratified by tiers

Word(k) Domain of words of tier k

$\mathbf{0}, \mathbf{1} : \mathbf{Word}(k) \rightarrow \mathbf{Word}(k)$ and $\epsilon : \mathbf{Word}(k)$

Word(1) Recursion

$\mathbf{0}, \mathbf{1} : \mathbf{Word}(1) \rightarrow \mathbf{Word}(1)$ and $\epsilon : \mathbf{Word}(1)$

Word(0) No recursion - flat domain

$\mathbf{0}, \mathbf{1} : \mathbf{Word}(0) \rightarrow \mathbf{Word}(0)$ and $\epsilon : \mathbf{Word}(0)$

Refer to the same set of words, $[\![\mathbf{Word}(k)]\!] = \{0, 1\}^*$

Tiered recursion

$$f(\epsilon, y) = g(y)$$
$$f(\mathbf{0}(x), y) = h_0(x, y, f(x, y))$$
$$f(\mathbf{1}(x), y) = h_1(x, y, f(x, y))$$

$$g : \mathbf{Word}(m) \to \mathbf{Word}(n)$$
$$h_i : \mathbf{Word}(n+1) \to \mathbf{Word}(m) \to \mathbf{Word}(n) \to \mathbf{Word}(n)$$
$$f : \mathbf{Word}(n+1), \mathbf{Word}(m) \to \mathbf{Word}(n)$$

Tier of Recurrence param. $>$ Tier of the recursive calls

Now the inputs and outputs have colors.

Keep that in mind !!!

Addition and Multiplication

Concatenation or addition :

$$\begin{aligned} \mathtt{add}(\epsilon, x) &= x \\ \mathtt{add}(\mathbf{0}(w), x) &= \mathbf{0}(\mathtt{add}(w, x)) \\ \mathtt{add}(\mathbf{1}(w), x) &= \mathbf{1}(\mathtt{add}(w, x)) \end{aligned}$$

$$\mathtt{add} : \mathbf{Word}(n+1) \to \mathbf{Word}(n) \to \mathbf{Word}(n)$$

Multiplication by iterating addition

$$\begin{aligned} \mathtt{mul}(\epsilon, y) &= \epsilon \\ \mathtt{mul}(\mathbf{0}(v), y) &= \mathtt{add}(y, \mathtt{mul}(v, y)) \\ \mathtt{mul}(\mathbf{1}(v), y) &= \mathtt{add}(y, \mathtt{mul}(v, y)) \end{aligned}$$

$$\mathtt{mul} : \mathbf{Word}(n+1) \to \mathbf{Word}(n+1) \to \mathbf{Word}(n)$$

Down casting

$$\texttt{cast}(\epsilon) = \epsilon$$
$$\texttt{cast}(\mathbf{0}(w)) = \mathbf{0}(\texttt{cast}(w))$$
$$\texttt{cast}(\mathbf{1}(w)) = \mathbf{1}(\texttt{cast}(w))$$

$$\texttt{cast} : \textbf{Word}(n+1) \to \textbf{Word}(n)$$

But up-casting is forbidden !
$\Rightarrow$ strict data ramification

$$\ldots > \textbf{Word}(k+1) > \textbf{Word}(k) > \ldots > \textbf{Word}(1) > \textbf{Word}(0)$$

Flat recurrence

Special case of tiered recursion where there is no recursive call

$$f(\epsilon, \overline{y}) = g(\overline{y})$$

$$g : \mathbf{Word}(n) \to \mathbf{Word}(n)$$

$$f(\mathbf{0}(x), \overline{y}) = h_0(x, \overline{y})$$

$$h_i : \mathbf{Word}(n) \to \mathbf{Word}(n) \to \mathbf{Word}(n)$$

$$f(\mathbf{1}(x)), \overline{y}) = h_1(x, \overline{y})$$

$$f : \mathbf{Word}(n) \to \mathbf{Word}(n) \to \mathbf{Word}(n)$$

$$\mathtt{pred}(\epsilon) = \epsilon \qquad \mathtt{pred}(\mathbf{0}(x), y) = x$$

$$\mathtt{pred}(\mathbf{1}(x), y) = x$$

$$\mathtt{cond}(\epsilon, y, z, w) = w \qquad \mathtt{cond}(\mathbf{0}(x), y, z, w) = y$$

$$\mathtt{cond}(\mathbf{1}(x), y, z, w) = z$$

Simultaneous tiered recursion

$$
\begin{aligned}
f_0(\epsilon,\overline{y}) = g_0(\overline{y}) \quad \ldots \quad f_p(\epsilon,\overline{y}) = g_p(\overline{y})\\
f_0(\mathbf{0}(x),\overline{y}) = h_0(x,\overline{y},f_0(x,\overline{y}),\ldots,f_p(x,\overline{y}))\\
f_0(\mathbf{1}(x),\overline{y}) = h'_0(x,\overline{y},f_0(x,\overline{y}),\ldots,f_p(x,\overline{y}))\\
\ldots\\
f_p(\mathbf{0}(x),\overline{y}) = h_0(x,\overline{y},f_0(x,\overline{y}),\ldots,f_p(x,\overline{y}))\\
f_p(\mathbf{1}(x),\overline{y}) = h'_0(x,\overline{y},f_0(x,\overline{y}),\ldots,f_p(x,\overline{y}))
\end{aligned}
$$

where

$$
\begin{aligned}
g_i : \mathbf{Word}(i) \rightarrow \mathbf{Word}(j)\\
h_i, h'_i : \mathbf{Word}(k+1) \rightarrow \mathbf{Word}(i) \rightarrow \mathbf{Word}(j)^p \rightarrow \mathbf{Word}(j)\\
f_i : \mathbf{Word}(k+1) \rightarrow \mathbf{Word}(i) \rightarrow \mathbf{Word}(j)
\end{aligned}
$$

Tiering condition implies

$$k+1 \geq i > j$$

Characterization of PTIME

Definition

The class $\mathbf{TRec}^*(\mathbf{Word})$ is the set of functions defined by simultaneous tiered recursion and explicit definitions (projections and composition well typed).

Theorem (Leivant 94)

The three sets are identical

- *The set* PTIME *of functions computable in polynomial time.*
- *The set* $\mathbf{TRec}^*(\boldsymbol{Word})$ *using any tiers*
- *The set* $\mathbf{TRec}^*(\boldsymbol{Word})$ *using 2 tiers only*

Proof.

We are going to sketch it shortly. $\square$

Another look at the exponential

Double the length

$$\begin{aligned}\mathtt{double}(\epsilon) &= \epsilon \\ \mathtt{double}(\mathbf{0}(w)) &= \mathbf{1}(\mathbf{1}(\mathtt{double}(w))) \\ \mathtt{double}(\mathbf{1}(w)) &= \mathbf{1}(\mathbf{1}(\mathtt{double}(w)))\end{aligned}$$

$$\mathtt{double} : \mathbf{Word}(n+1) \to \mathbf{Word}(n)$$

Exponential by doubling

$$\begin{aligned}\mathtt{exp}(\epsilon) &= \mathbf{1}(\epsilon) \\ \mathtt{exp}(\mathbf{0}(w)) &= \mathtt{double}(\mathtt{exp}(w)) \\ \mathtt{exp}(\mathbf{1}(w)) &= \mathtt{double}(\mathtt{exp}(w))\end{aligned}$$

$$\mathtt{exp} : \mathbf{Word}(k+1) \to \mathbf{Word}(k)$$

No solution, this definition is circular !!!

Tiered recursion captures PTIME

Take a Turing Machine M,

- q is a state
- u the left tape
- v the right tape
- the head is scanning the first letter of u

$$\begin{aligned}\texttt{state}(q,u,v) &= \text{next state}\\ \texttt{left}(q,u,v) &= \text{left side of the tape}\\ \texttt{right}(q,u,v) &= \text{right side of the tape}\end{aligned}$$

build by explicit definitions

$$\texttt{state}, \texttt{left}, \texttt{right} : \textbf{Word}(0)^3 \rightarrow \textbf{Word}(0)$$

Linear length Iteration

$$
\begin{aligned}
T_1^S(\epsilon, q, u, v) &= q \\
T_1^L(\epsilon, q, u, v) &= u \\
T_1^R(\epsilon, q, u, v) &= v \\
T_1^S(\mathbf{i}(t), q, u, v) &= \mathtt{state}(T_1^S(t, q, u, v), \\
&\qquad T_1^L(t, q, u, v), T_1^R(t, q, u, v)) \\
T_1^L(\mathbf{i}(t), q, u, v) &= \mathtt{left}(T_1^S(t, q, u, v), \\
&\qquad T_1^L(t, q, u, v), T_1^R(t, q, u, v)) \\
T_1^R(\mathbf{i}(t), q, u, v) &= \mathtt{right}(T_1^S(t, q, u, v), \\
&\qquad T_1^L(t, q, u, v), T_1^R(t, q, u, v))
\end{aligned}
$$

Linear length Iteration

$$T_1^S, T_1^L, T_1^R : \mathbf{Word}(1) \to \mathbf{Word}(0)^3 \to \mathbf{Word}(0)$$

$$\begin{aligned}
[\![T_1^S(t,q,u,v)]\!] &= \text{state after } t \text{ steps} \\
[\![T_1^L(t,q,u,v)]\!] &= \text{left tape after } t \text{ steps} \\
[\![T_1^R(t,q,u,v)]\!] &= \text{right tape after } t \text{ steps}
\end{aligned}$$

We make k nested simultaneous recursion to iterate n^k times

Polynomial length iteration

$$T^S_{k+1}(\epsilon, q, u, v) = q \quad T^L_{k+1}(\epsilon, q, u, v) = u \quad T^R_{k+1}(\epsilon, q, u, v) = v$$

$$\begin{aligned}
T^S_{k+1}(\mathbf{i}(t), \bar{t}, q, u, v) &= T^S_k(\bar{t}, T^S_k(\bar{t}, q, u, v,), \\
&\qquad T^L_k(\bar{t}, q, u, v), T^R_k(\bar{t}, q, u, v)) \\
T^L_{k+1}(\mathbf{i}(t), \bar{t}, q, u, v) &= T^L_k(\bar{t}, T^S_k(\bar{t}, q, u, v,), \\
&\qquad T^L_k(\bar{t}, q, u, v), T^R_k(\bar{t}, q, u, v)) \\
T^R_{k+1}(\mathbf{i}(t), \bar{t}, q, u, v) &= T^R_k(\bar{t}, T^S_k(\bar{t}, q, u, v,), \\
&\qquad T^L_k(\bar{t}, q, u, v), T^R_k(\bar{t}, q, u, v))
\end{aligned}$$

where $\bar{t} = t, t_k, \ldots, t_1$

$$T^S_{k+1}, T^L_{k+1}, T^R_{k+1} : \mathbf{Word}(1)^{k+1} \to \mathbf{Word}(0)^3 \to \mathbf{Word}(0)$$

Simulation of PTIME computation

Lemma

A polynomial time computable function $\phi : \{0,1\}^* \to \{0,1\}^*$ *is captured by a function in* **TRec***(***Word***) *using 2 tiers only*

Proof.

Suppose that Φ is computed by a TM in time n^k.

$$\phi(w) = T_k^R(w, \ldots, w, q_0, w, \epsilon)$$

where

$$[\![T_k^S(\overline{w}, q, u, v)]\!] = \text{state after } |w|^k \text{ steps}$$
$$[\![T_k^L(\overline{w}, q, u, v)]\!] = \text{left tape after } |w|^k \text{ steps}$$
$$[\![T_k^R(\overline{w}, q, u, v)]\!] = \text{right tape after } |w|^k \text{ steps}$$

$\square$

Computation of tiered recursion

Lemma

For any tier 1 arguments $u_1, \ldots, u_p$, and tier 0 arguments $v_1, \ldots, v_q$, the computation of $f(u_1, \ldots, u_p, v_1, \ldots, v_q)$ runs in time bounded by $c \times (\sum_{i=1,p} |u_i|)^k$.

Proof

$$
\begin{aligned}
f(\epsilon, y; z) &= g(y; z) \\
f(\mathbf{0}(x), y; z) &= h_0(x, y; z, f(x, y; z)) \\
f(\mathbf{1}(x), y; z) &= h_1(x, y; z, f(x, y; z))
\end{aligned}
$$

$$f : \mathbf{Word}(1), \mathbf{Word}(1), \mathbf{Word}(0) \to \mathbf{Word}(0)$$

- Ind. Hyp applies to g and h_i
- So, g runs within a time bounded by a polynomial in tier 1 inputs

$$Time(g(v; w)) \leq |v|^{k'}$$

- So, the run time of h_i is polynomial in tier 1 inputs

$$Time(h_i(u', v; w, w')) \leq (|u'| + |v|)^{k''}$$

- $f(u, v, w)$ is computed by iterating $|u|$ times h_i's

$$
\begin{aligned}
Time(f(u, v, w)) &\leq |u| \times (|u| + |v|)^{k''} + |v|^{k'} \\
&\leq (|u| + |v|)^{k' + k''}
\end{aligned}
$$

Church-numerals

$$\underline{n} = \lambda f : \alpha \to \alpha\, \lambda x : \alpha, f^{n}(x)$$

where $f^{0}(x) = x$ and $f^{k+1}(x) = f(f^{k}(x))$

$$\underline{n} : \mathbf{N}(\alpha) = (\alpha \to \alpha) \to (\alpha \to \alpha)$$

Successor :

$$\mathbf{suc} = \lambda n : \mathbf{N}(\alpha)\ \lambda f : \alpha \to \alpha\ \lambda x : \alpha,\ f(n\ f\ x)$$

$$\underline{0} = \lambda f\, \lambda x, x \qquad \underline{n+1} = (\mathbf{suc}\ \underline{n})$$

Church numeral arithmetic

$$(\texttt{add}\ \underline{n}\ \underline{m}) = \lambda f \lambda x, (\underline{n}\ f\ (\underline{m}\ f\ x))$$
$$\texttt{add} : \mathbf{N}(\alpha) \rightarrow \mathbf{N}(\alpha) \rightarrow \mathbf{N}(\alpha)$$
$$[\![\texttt{add}]\!](\underline{n}, \underline{m}) = \underline{n + m}$$

$$(\texttt{mul}\ \underline{n}\ \underline{m}) = \lambda f \lambda x, (\underline{n}\ (\underline{m}\ f)\ x)$$
$$\texttt{mul} : \mathbf{N}(\alpha) \rightarrow \mathbf{N}(\alpha) \rightarrow \mathbf{N}(\alpha)$$
$$[\![\texttt{mul}]\!](\underline{n}, \underline{m}) = \underline{n \times m}$$

But

$$(\texttt{exp}\ \underline{n}) = (\underline{n}\ \underline{2})$$
$$\texttt{exp} : \mathbf{N}(\alpha \rightarrow \alpha) \rightarrow \mathbf{N}(\alpha)$$
$$[\![\texttt{exp}]\!](\underline{n}) = \underline{2^n}$$

Extended polynomials

Definition

A function $\phi : \mathbb{N} \to \mathbb{N}$ is Church-representable if there is a λ-term F such that

$$(F \underline{n}) = \underline{\phi(n)}$$
$$F : \mathbf{N}(\alpha) \to \mathbf{N}(\alpha)$$

- Exponential is not Church-representable

Theorem (Schwichtenberg)

The set of Church-representable functions is the set of extended polynomials (= polynomials + test if $n = 0$).

Church representation of first order structures

$$\mathbf{W}(\alpha) = (\alpha \to \alpha) \to (\alpha \to \alpha) \to (\alpha \to \alpha)$$

$$\begin{aligned} \underline{\epsilon} &= \lambda f_0 \lambda f_1 \lambda x, x \\ \underline{1} &= \lambda u \lambda f_0 \lambda f_1 \lambda x, f_0(\underline{u} f_0 f_1 x) \\ \underline{0} &= \lambda u \lambda f_0 \lambda f_1 \lambda x, f_1(\underline{u} f_0 f_1 x) \end{aligned}$$

$$\underline{\mathbf{0}(u)} = (\underline{0}\ \underline{u}) \qquad \underline{\mathbf{1}(u)} = (\underline{1}\ \underline{u})$$

$$\mathbf{W}(\alpha) = (\alpha \to \alpha) \to (\alpha \to \alpha) \to (\alpha \to \alpha)$$
$$\underline{u} : \mathbf{W}(\alpha)$$

Böhm and Berarducci (85)

Two levels of data representations

Abstract level : $\lambda u \lambda f_0 \lambda f_1 \lambda x \, , f_0(f_0(f_1 x))$

Data level : $\mathbf{0}(\mathbf{0}(\mathbf{1}(\epsilon)))$

- Abstract level $\Rightarrow$ Data Level
- But the converse does not hold
- Data ramification principle !

λ-calculus over **Word***

Atomic types : **Word** and **Word***

Constructors :

$$\epsilon : \textbf{Word} \qquad \textbf{0}, \textbf{1} : \textbf{Word} \to \textbf{Word}$$

$$\textbf{nil} : \textbf{Word}^* \qquad \textbf{cons} : \textbf{Word} \to \textbf{Word}^* \to \textbf{Word}^*$$

λ-calculus over **Word***

Destructors:

$$\mathtt{pred}(\epsilon) = \epsilon \qquad \mathtt{pred}(\mathbf{0}(u)) = u \qquad \mathtt{pred}(\mathbf{1}(u)) = u$$

$$\mathtt{cond}(u, a, b, c) = \begin{cases} a & u = \epsilon \\ b & u = \mathbf{0}(u) \\ c & u = \mathbf{1}(u) \end{cases}$$

$$\mathrm{hd}(\mathbf{nil}) = \mathbf{nil} \qquad \mathrm{hd}(\mathbf{cons}(u, L)) = u$$

$$\mathrm{tl}(\mathbf{nil}) = \mathbf{nil} \qquad \mathrm{tl}(\mathbf{cons}(u, L)) = L$$

$$\mathtt{cond}(L, a, b) = \begin{cases} a & u = \mathbf{nil} \\ b & \text{otherwise} \end{cases}$$

Church representation of algebra terms

Definition

$\mathbf{1}\lambda^{\mathbf{p}}(\mathbf{Word}^*)$ is the class of terms of simply typed λ-calculus with constructors and destructors over $\mathbf{Word}^*$

Definition

A function $\Phi : \{0,1\}^* \to \{0,1\}^*$ is computed by a λ-term F of $\mathbf{1}\lambda^{\mathbf{p}}(\mathbf{Word}^*)$ if $F(\underline{w}) = \phi(w)$ where $\underline{f} : \mathbf{W}(\mathbf{Word}^*) \to \mathbf{Word}^*$.

Theorem (Leivant-Marion)

The set of functions computed by $\mathbf{1}\lambda^{\mathbf{p}}(\boldsymbol{Word}^*)$*-terms is exactly the set* PTIME *of polynomial time functions.*

References

S. Bellantoni and S. Cook.
A new recursion-theoretic characterization of the poly-time functions.
Computational Complexity, 2:97–110, 1992.

D. Leivant.
A foundational delineation of poly-time.
Information and Computation, 110(2):391–420, 1994.

D. Leivant and J-Y Marion.
Lambda calculus characterizations of poly-time.
Fundamenta Informaticae, 19(1,2):167,184, September 1993.

H. Simmons.
The realm of primitive recursion.
Archive for Mathematical Logic, 27:177–188, 1988.

Parallel Register Machines (PRM)

1. a finite set $S = \{s_0, s_1, \ldots, s_k\}$ of *states*
2. a finite list $\Pi = \{\pi_1, \ldots, \pi_m\}$ of *registers*
3. and commands
   - [$\mathbf{Succ}(\pi = \mathbf{i}(\pi), s')$], [$\mathbf{Pred}(\pi = \mathbf{p}(\pi), s')$],
   - [$\mathbf{Branch}(\pi, s', s'')$],
   - [$\mathbf{Fork}_{\min}(s', s'')$], [$\mathbf{Fork}_{\max}(s', s'')$],
   - [**End**].

(PRM)

$$\texttt{eval}(0, s, \Pi) = \perp$$

$\texttt{cmd}(s) = \pi_j = \mathbf{i}(\pi_j)$ and the next state is s'

$$\texttt{eval}(t+1, s, \Pi) = \texttt{eval}(t, s', \{\pi \leftarrow \mathbf{i}(\pi)\}\Pi)$$

$\texttt{cmd}(s) = \texttt{pred}(\pi)$

$$\texttt{eval}(t+1, s, \Pi) = \texttt{eval}(t, s', \{\pi \leftarrow \mathbf{p}(\pi)\}\Pi)$$

$\texttt{cmd}(s) = \mathbf{Branch}(\pi, s', s'')$

$$\texttt{eval}(t+1, s, \Pi) = \begin{cases} \texttt{eval}(t, s', \Pi) & \text{if } \pi = \mathbf{0}(w) \\ \texttt{eval}(t, s'', \Pi) & \text{if } \pi = \mathbf{1}(w) \end{cases}$$

PRM

And the Fork

$\texttt{cmd}(s) = \textbf{Fork}_{\min}(s', s'')$

$$\texttt{eval}(t+1, s, \Pi) = \min_{\blacktriangleleft}(\texttt{eval}(t, s', \Pi), \texttt{eval}(t, s'', \Pi))$$

$\texttt{cmd}(s) = \textbf{Fork}_{\max}(s', s'')$

$$\texttt{eval}(t+1, s', \Pi) = \max_{\blacktriangleleft}(\texttt{eval}(t, s', \Pi), \texttt{eval}(t, s'', \Pi))$$

where $\blacktriangleleft$ is the lexicographic order on words.

$$\texttt{eval}(t+1, \textbf{End}, \Pi) = \Pi(\textsc{output})$$

PRM

A function $\phi : \{0,1\}^* \to \{0,1\}^*$ is PRM-computable in time $T : \mathbb{N} \to \mathbb{N}$ if there is a PRM M such that for each $(w_1, \cdots, w_k) \in \mathbb{W}^k$, we have

$$\mathtt{eval}(T(|w|), \mathrm{BEGIN}, F_0) = \phi(w)$$

Time-bound semantics

$$\mathtt{eval} : \mathbb{N} \times S \times \mathbb{W}^m \mapsto \mathbb{W}$$

Trade-off between time and space

Theorem

The following are equivalent

1. ϕ *is computable in polynomial space*
2. ϕ *is computable in non-deterministic polynomial space*
3. ϕ *is computable in polynomial time on Alternating Turing Machine*
4. ϕ *is computable in polynomial time on PRM*

Proof.

See Chandra, Kozen, Stockmeyer and Savitch □

Tiered recursion with substitutions

$$
\begin{aligned}
f(\epsilon, \overline{y}) &= g(\overline{y}) \\
f(\mathbf{0}(x), \overline{y}) &= h_0(x, \overline{y}, f(x, \sigma_1(\overline{y})), \ldots, f(x, \sigma_k(\overline{y}))) \\
f(\mathbf{1}(x), \overline{y}) &= h_1(x, \overline{y}, f(x, \sigma'_1(\overline{y})), \ldots, f(x, \sigma'_k(\overline{y})))
\end{aligned}
$$

$$\sigma_i, \sigma'_i : \mathbf{Word}(m) \to \mathbf{Word}(n)$$
$$g : \mathbf{Word}(m) \to \mathbf{Word}(n)$$
$$h_i : \mathbf{Word}(n+1) \to \mathbf{Word}(m) \to \mathbf{Word}(n) \to \mathbf{Word}(n)$$
$$\sigma_j, \sigma' j : \mathbf{Word}(m) \to \mathbf{Word}(m)$$
$$f : \mathbf{Word}(n+1), \mathbf{Word}(m) \to \mathbf{Word}(n)$$

Characterization of PSPACE

Definition

The class **Sub**(**Word**) is the set of functions defined by tiered recursion with substitutions and explicit definitions (projections and composition).

Theorem (LM95)

The three sets are identical

- *The set* PSPACE *of functions computable in polynomial space*
- *The set* **Sub**(***Word***) *using any tiers*
- *The set* **Sub**(***Word***) *using 3 tiers only*

Proof.

We are going to sketch it shortly. □

Simulating PSPACE

Lemma

A polynomial time PRM computable function $\phi : \{0,1\}^* \to \{0,1\}^*$ *is captured by a function in* $\mathbf{Sub}(\boldsymbol{Word})$ *using 3 tiers only*

Proof.

`eval` is defined by rec. with substitution of parameters:
$\mathtt{cmd}(s) = \mathbf{Fork}_{\min}(s', s'')$

$$\mathtt{eval}(t+1, s, \Pi) = \min(\mathtt{eval}(t, \delta_0(s), \Pi), \mathtt{eval}(t, \delta_1(s), \Pi))$$

where

$$\delta_0(s) = s'$$
$$\delta_1(s) = s''$$

$$\mathtt{eval} : \mathbf{Word}(1) \to \mathbf{Word}(0)^m \to \mathbf{Word}(0)$$

□

Simulating PSPACE

Define a polynomial time clock $T : \textbf{Word}(2) \to \textbf{Word}(1)$ by composing tiered addition and multiplication that we have already seen.

$$\phi(w) = \mathtt{eval}(T(|w|), \text{BEGIN}, F_0)$$

Computation in PSPACE

Lemma

A function ϕ in **Sub(*Word*)** *using 3 tiers only is computed in space $O(n^k) + m$ for some k*

- *n is the size of tier 2 and 1 arguments*
- *m is the size of tier 0 arguments*

Other tiered characterizations of low complexity classes

- $NC^1 \equiv$ A-LOG-TIME
  - *Bloch (94)*,
  - *Leivant-Marion (00)*
- NC^k
  - *Bonfante, Kähle, Marion, Oitavem (06)*,
- NC
  - *Leivant (98)*,
  - *Oitavem (04)*
- NP
  - *Bellantoni (94)*
- FPSPACE $\equiv$ A-POLY-TIME
  - *Leivant-Marion (95)*

Computing over a structure $\mathcal{K}$

A computational structure

$$\mathcal{K} = \langle \mathbb{K}, \{op_i\}_{i \in I}, rel_1 \ldots, rel_\ell, =, \mathbf{0}, \mathbf{1} \rangle$$

- A domain $\mathbb{K}$
- operators $\{op_i\}_{i \in I}$ over $\mathbb{K}$
- relations $rel_1, \ldots, rel_l$
- the equality over $\mathbb{K}$
- two particular constants $\mathbf{0}$ and $\mathbf{1}$

$\mathbb{K}^*$ denotes lists of elements of $\mathbb{K}$.

Reference

S. Bellantoni.
Predicative recursion and the polytime hierarchy.
In Peter Clote and Jeffery Remmel, editors, *Feasible Mathematics II*, *Perspectives in Computer Science*. Birkhäuser, 1994.

S. Bloch.
Function-algebraic characterizations of log and polylog parallel time.
Computational complexity, 4(2):175–205, 1994.

D. Leivant.
A characterization of NC by tree recurrence.
In *39th Annual Symposium on Foundations of Computer Science, FOCS'98*, pages 716–724, 1998.

D. Leivant and J-Y Marion.
Ramified recurrence and computational complexity II: substitution and poly-space.
In L. Pacholski and J. Tiuryn, editors, *Computer*

Blum-Shub-Smale machine over $\mathcal{K}$

Similar to a Turing machine, with the properties:

- Its tape cells hold arbitrary elements of $\mathbb{K}$.
- It has Computation nodes for computing the operations $\{op_i\}_{i \in I}$ with unit cost.
- It has Branch nodes for computing the relations $rel_1, \ldots, rel_l$ with unit cost.
- It has Shift nodes for moving the head.
- Inputs and outputs are vectors in $\mathbb{K}^*$

A TM computes a function from $\mathbb{K}^*$ to $\mathbb{K}^*$.

$\mathbb{K}^*$ denotes lists of elements of $\mathbb{K}$.

Polynomial time functions over $\mathcal{K}$

Definition

A function $f : (\mathbb{K}^*)^n \rightarrow (\mathbb{K}^*)^m$ is in class $\mathrm{PTIME}_{\mathcal{K}}$
iff
f is computable in polynomial time.
That is,
there is a polynomial p and a BSS-TM M, such that

- M computes f
- M stops in $p(|\overline{w}|)$ steps on each input w of $\mathbb{K}^*$.

$|\overline{w}|$ is the length of the list $\overline{w} \in \mathbb{K}^*$.

Complexity Theory over $\mathcal{K}$

- Over the structure $\mathcal{B} = (\{0,1\}, =, 0, 1)$, we compute $f : \{0,1\}^* \rightarrow \{0,1\}^*$, corresponds to classical complexity and $\mathrm{PTIME}_{\mathcal{B}} = \mathrm{PTIME}$.
- Over the structure $\mathcal{R} = (\mathcal{R}, +, -, *, /, >, =, 0, 1)$ corresponds to the original Blum Shub and Smale (89) paper.

Safe Recursion over a structure $\mathcal{K}$

Two types of arguments,“normal”and “safe”

$$f(\overline{x}; \overline{y})$$

The set of safe recursive functions over $\mathcal{K}$ is the smallest set of functions containing basic safe functions

- structure operators and relations
- projections
- list destructors : **hd** and **tl**
- list constructor : **cons**
- Boolean selection : **if** $x = 1$ **then** y **else** z

Basic functions

- $\mathrm{hd}(;a.\overline{x}) = a,\ \mathrm{tl}(;a.\overline{x}) = \overline{x},\ \mathrm{cons}(;a.\overline{x},\overline{y}) = a.\overline{y}$
- Projections
- Application of operators and relations

$$\begin{aligned} \mathrm{Op}_\imath(;a_1.\overline{x_1},\ldots,a_{n_\imath}.\overline{x_{n_\imath}}) &= (op_\imath(a_1,\ldots,a_{n_\imath})).\overline{x_{n_\imath}} \\ \mathrm{Rel}_\imath(;a_1.\overline{x_1},\ldots,a_{n_\imath}.\overline{x_{n_\imath}}) &= \begin{cases} \mathbf{1} \text{ if } rel_\imath(a_1,\ldots,a_{n_\imath}) \\ \epsilon \text{ otherwise} \end{cases} \end{aligned}$$

- Test

$$\mathrm{Select}(;\overline{x},\overline{y},\overline{z}) = \begin{cases} \overline{y} & \text{if } \mathrm{hd}(\overline{x}) = \mathbf{1} \\ \overline{z} & \text{otherwise} \end{cases}$$

Safe Recursive functions over $\mathcal{K}$

and closed under both schemas

- safe composition

$$f(\overline{x};\overline{y}) = g(h_1(\overline{x};);h_2(\overline{x};\overline{y}))$$

- safe recursion

$$f(\epsilon,\overline{x};\overline{y}) = g(\overline{x};\overline{y})$$
$$f(a.\overline{z},\overline{x};\overline{y}) = g(\overline{z},\overline{x};f(\overline{z},\overline{x};\overline{y}),\overline{y})$$

Polynomial time functions PTIME$_{\mathcal{K}}$

Theorem (Bournez-Cucker-de Naurois-Marion (03))

Over any structure $\mathcal{K}$, the set of safe recursive functions over $\mathcal{K}$ is exactly PTIME$_{\mathcal{K}}$.

Proof.

This proof implies Bellantoni and Cook's one and is more direct. □

About space

A priori, there is no valid notion of space over arbitrary structures.

Theorem (Michaux)

Over $(\mathbb{R}^+, 0, 1, =, +, -, *, <)$, *any computable function can be computed in constant working space.*

But, Paulin de Naurois gives a logarihmic cost, see his talk at the ICC workshop next week !

The class FPAR$_{\mathcal{K}}$

FPAR$_{\mathcal{K}}$ is the set of functions computable in parallel poly-time.
That is, by a P-uniform family of circuits of polynomial depth.

Theorem (Bournez-Cucker-deNaurois-Marion (04))

A function: $\mathbb{K}^* \to \mathbb{K}^*$ is computed in FPAR$_{\mathcal{K}}$ *if and only if it is defined as a safe recursive function with substitutions over $\mathcal{K}$.*

What we've seen

- Data Ramification Principle
  1. Normal/Safe recursions
  2. Tiering
  3. Simply typed λ-calculus
- Characterizations of PTIME and PSPACE
- Capture Turing Machine over arbitrary structures

References

O. Bournez, F. Cucker, P.J. de Naurois, and J.Y. Marion.
Implicit complexity over an arbitrary structure: Quantifier alternations.
Information and Computation, 204(2):210–230, Feb 2006.

Olivier Bournez, Felipe Cucker, Paulin Jacobé de Naurois, and Jean-Yves Marion.
Implicit complexity over an arbitrary structure: Sequential and parallel polynomial time.
Journal of Logic and Computation, 15(1):41–58, 2005.

Paulin Jacobé de Naurois.
Résultats de Complétude et Caractérisations Syntaxiques de Classes de Complexité sur des Structures Arbitraires.
PhD thesis, INPL and City university (Hong-Kong), Dec 2004.

Conclusion

- Intrinsic characterizations
  - "resource" is inside Data Ramification Principle
  - Syntactic complexity characterization
    - we may extract bound,
    - but, low algorithmic expressiveness
  - quite robust wrt model of computations
- Can we apply data ramification to other models of computation ?
- Studying intentional characterization of complexity classes.
- Developing automatic resource analysis by mean of static analysis.