Algorithms and implementations

Yiannis N. Moschovakis
UCLA and University of Athens

Tarski Lecture 1, March 3, 2008

What is an algorithm?

- Basic aim: to "define" (or represent) algorithms in set theory, in the same way that we represent *real numbers* (Cantor, Dedekind) and *random variables* (Kolmogorov) by set-theoretic objects
- What set-theoretic objects represent algorithms?
- When do two two set-theoretic objects represent the same algorithm? (The algorithm identity problem)
- In what way are algorithms effective?
- ...and do it so that the basic results about algorithms can be established rigorously (and naturally)
- ...and there should be some applications!

Plan for the lectures

Lecture 1. *Algorithms and implementations*
Discuss the problem and some ideas for solving it

Lecture 2. *English as a programming language*
Applications to Philosophy of language (and linguistics?)
synonymy and faithful translation $\sim$ algorithm identity

Lecture 3. *The axiomatic derivation of absolute lower bounds*
Applications to complexity (joint work with Lou van den Dries)
Do not depend on pinning down algorithm identity

Lectures 2 and 3 are independent of each other and mostly independent of Lecture 1

I will oversimplify, but: All lies are white (John Steel)

Outline of Lecture 1

Slogan: *The theory of algorithms is the theory of recursive equations*

(1) Three examples
(2) Machines vs. recursive definitions
(3) Recursors
(4) Elementary algorithms
(5) Implementations

Notation:

$$\mathbb{N} = \{0, 1, 2, \ldots\}$$

$$a \geq b \geq 1, \quad a = bq + r, \quad 0 \leq r < b$$
$$\implies q = \mathrm{iq}(a, b), \ r = \mathrm{rem}(a, b)$$

$$\gcd(a, b) = \text{the greatest common divisor of } a \text{ and } b$$

$$a \perp\!\!\!\perp b \iff \mathrm{rem}(a, b) = 1 \quad (a \text{ and } b \text{ are coprime})$$

The Euclidean algorithm ε

For $a, b \in \mathbb{N}, a \geq b \geq 1$,

$$\varepsilon : \boxed{\gcd(a,b) = \text{if } (\text{rem}(a,b) = 0) \text{ then } b \text{ else } \gcd(b, \text{rem}(a,b))}$$

$c_\varepsilon(a,b) =$ the number of divisions needed to compute $\gcd(a,b)$ using ε

Complexity of the Euclidean

If $a \geq b \geq 2$, *then* $c_\varepsilon(a,b) \leq 2\log_2(a)$

Proofs of the correctness and the upper bound are by induction on $\max(a,b)$

What is the Euclidean algorithm?

$$\varepsilon : \boxed{\gcd(a,b) = \text{if } (\text{rem}(a,b) = 0) \text{ then } b \text{ else } \gcd(b, \text{rem}(a,b))}$$

- It is an algorithm on $\mathbb{N}$, from (relative to) the remainder function rem and it computes $\gcd : \mathbb{N}^2 \to \mathbb{N}$
- It is needed to make precise the optimality of the Euclidean:

Basic Conjecture

For every algorithm α which computes on $\mathbb{N}$ from rem *the greatest common divisor function, there is a constant $r > 0$ such that for infinitely many pairs $a \geq b \geq 1$,*

$$c_\alpha(a,b) \geq r \log_2(a)$$

Sorting (alphabetizing)

Given an ordering $\leq$ on a set A and any $u = \langle u_0, \ldots, u_{n-1} \rangle \in A^n$

$$\mathrm{sort}(u) = \text{the unique, sorted (non-decreasing) rearrangement } v = \langle u_{\pi(0)}, u_{\pi(1)}, \ldots, u_{\pi(n-1)} \rangle$$

where $\pi : \{0, \ldots, n-1\} \rightarrowtail\!\!\!\!\twoheadrightarrow \{0, \ldots, n-1\}$ is a permutation

$$\begin{aligned}
\mathrm{head}(\langle u_0, \ldots, u_{n-1} \rangle) &= u_0 \\
\mathrm{tail}(\langle u_0, \ldots, u_{n-1} \rangle) &= \langle u_1, \ldots, u_{n-1} \rangle \\
\langle x \rangle * \langle u_0, \ldots, u_{n-1} \rangle &= \langle x, u_0, \ldots, u_{n-1} \rangle \quad \text{(prepend)} \\
|\langle u_0, \ldots, u_{n-1} \rangle| &= n \quad \text{(the length of } u\text{)} \\
h_1(u) &= \text{the first half of } u \quad \text{(the first half)} \\
h_2(u) &= \text{the second half of } u \quad \text{(the second half)}
\end{aligned}$$

The mergesort algorithm σ_m

$$\mathrm{sort}(u) = \text{if } (|u| \leq 1) \text{ then } u \text{ else } \mathrm{merge}(\mathrm{sort}(h_1(u)), \mathrm{sort}(h_2(u)))$$

$$\mathrm{merge}(v, w) = \begin{cases} w & \text{if } |v| = 0, \\ v & \text{else, if } |w| = 0, \\ \langle v_0 \rangle * \mathrm{merge}(\mathrm{tail}(v), w) & \text{else, if } v_0 \leq w_0, \\ \langle w_0 \rangle * \mathrm{merge}(v, \mathrm{tail}(w)) & \text{otherwise.} \end{cases}$$

(1) If v, w are sorted, then $\mathrm{merge}(v, w) = \mathrm{sort}(w * v)$

(2) The sorting and merging function satisfy these equations

(3) $\mathrm{merge}(v, w)$ can be computed using no more than $|v| + |w| \dot{-} 1$ comparisons

(4) $\mathrm{sort}(u)$ can be computed by σ_m using no more than $|u| \log_2(|u|)$ comparisons ($|u| > 1$)

What is the mergesort algorithm?

$$\mathrm{sort}(u) = \text{if } (|u| \leq 1) \text{ then } u \text{ else } \mathrm{merge}(\mathrm{sort}(h_1(u)), \mathrm{sort}(h_2(u)))$$

$$\mathrm{merge}(v, w) = \begin{cases} w & \text{if } |v| = 0, \\ v & \text{else, if } |w| = 0, \\ \langle v_0 \rangle * \mathrm{merge}(\mathrm{tail}(v), w) & \text{else, if } v_0 \leq w_0, \\ \langle w_0 \rangle * \mathrm{merge}(v, \mathrm{tail}(w)) & \text{otherwise.} \end{cases}$$

$$c_{\sigma_m}(u) = \text{the number of comparisons needed to compute } \mathrm{sort}(u)$$
$$\text{using } \sigma_m \leq |u| \log_2(|u|) \quad (|u| > 0)$$

- It is an algorithm from the ordering $\leq$ and the functions $\mathrm{head}(u), \mathrm{tail}(u), |u|, \ldots$
- It is needed to make precise the optimality of σ_m:
  For every sorting algorithm σ from $\leq, \mathrm{head}, \mathrm{tail}, \ldots$, there is an $r > 0$ and infinitely many sequences u such that
  $c_\sigma(u) \geq r|u| \log_2(|u|)$ (well known)

The Gentzen Cut Elimination algorithm

Every proof d of the Gentzen system for Predicate Logic can be transformed into a cut-free proof $\gamma(d)$ with the same conclusion

$$\begin{aligned}\gamma(d) = \ & \text{if } T_1(d) \text{ then } f_1(d) \\ & \text{else if } T_2(d) \text{ then } f_2(\gamma(\tau(d))) \\ & \text{else } f_3(\gamma(\sigma_1(d)), \gamma(\sigma_2(d)))\end{aligned}$$

- It is a recursive algorithm from natural syntactic primitives, very similar in logical structure to the mergesort
- **Main Fact**: $|\gamma(d)| \leq e(\rho(d), |d|)$, where $|d|$ is the length of the proof d, $\rho(d)$ is its cut-rank, and

$$e(0, k) = k, \quad e(n+1, k) = 2^{e(n,k)}$$

The infinitary Gentzen algorithm

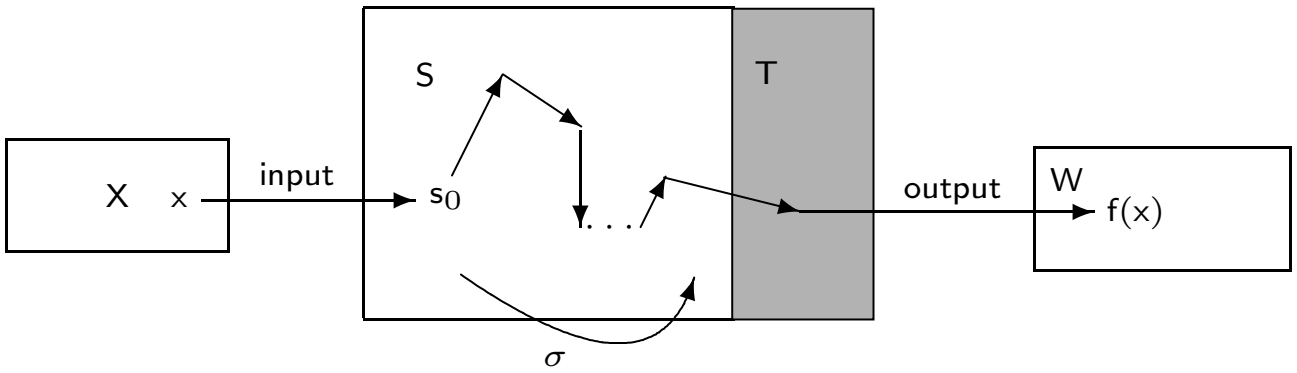
If we add the ω-rule to the Gentzen system for Peano arithmetic, then cuts can again be eliminated by an extension of the finitary Gentzen algorithm

$$\begin{aligned}\gamma^*(d) = \text{if } & T_1(d) \text{ then } f_1(d)\\ & \text{else if } T_2(d) \text{ then } f_2(\gamma^*(\tau(d)))\\ & \text{else if } T_3(d) \text{ then } f_3(\gamma^*(\sigma_1(d)), \gamma^*(\sigma_2(d)))\\ & \text{else } f_4(\lambda(n)\gamma^*(\rho(n,d))),\end{aligned}$$

where f_4 is a functional embodying the ω-rule

- Again $|\gamma^*(d)| \leq e(\rho(d), |d|)$, where cut-ranks and lengths of infinite proofs are ordinals, $e(\alpha, \beta)$ is defined by ordinal recursion, and so every provable sentence has a cut-free proof of length less than
  $\varepsilon_0 =$ the least ordinal > 0 and closed under $\alpha \mapsto \omega^\alpha$

Abstract machines (computation models)



A machine $\mathfrak{m} : X \rightsquigarrow Y$ is a tuple $(S, \text{input}, \sigma, T, \text{output})$ such that

1. S is a non-empty set (of states)
2. $\text{input} : X \to S$ is the input function
3. $\sigma : S \to S$ is the transition function
4. T is the set of terminal states, $T \subseteq S$
5. $\text{output} : T \to Y$ is the output function

$$\overline{\mathfrak{m}}(x) = \text{output}(\sigma^n(\text{input}(x)))$$

where $n =$ least such that $\sigma^n(\text{input}(x)) \in T$

Infinitary algorithms are not machines

- It is useful to think of the infinitary Gentzen "effective procedure" as an algorithm
- There are applications of infinitary algorithms (in Lecture 2)
- Machines are special algorithms which implement finitary algorithms
- The relation between an (implementable) algorithm and its implementations is interesting

Which machine is the Euclidean?

$$\varepsilon : \boxed{\gcd(a, b) = \text{if } (\text{rem}(a, b) = 0) \text{ then } b \text{ else } \gcd(b, \text{rem}(a, b))}$$

- Must specify a set of states, an input function, a transition function, etc.
- This can be done, in many ways, generally called implementations of the Euclidean
- The choice of a "natural" (abstract) implementation is irrelevant for the correctness and the log upper bound of the Euclidean, which are derived directly from the recursive equation above and apply to all implementations

- Claim: ε is completely specified by the equation above

Which machine is the mergesort algorithm?

$$\mathrm{sort}(u) = \text{if } (|u| \leq 1) \text{ then } u \text{ else } \mathrm{merge}(\mathrm{sort}(h_1(u)), \mathrm{sort}(h_2(u)))$$

$$\mathrm{merge}(v, w) = \begin{cases} w & \text{if } |v| = 0, \\ v & \text{else, if } |w| = 0, \\ \langle v_0 \rangle * \mathrm{merge}(\mathrm{tail}(v), w) & \text{else, if } v_0 \leq w_0, \\ \langle w_0 \rangle * \mathrm{merge}(v, \mathrm{tail}(w)) & \text{otherwise.} \end{cases}$$

- Many (essentially) different implementations
  sequential (with specified orders of evaluation), parallel, ...
- The correctness and $n \log_2(n)$ upper bound are derived directly from a (specific reading) of these recursive equations
- They should apply to all implementations of the mergesort
- Claim: σ_m is completely specified by the system above
- *Task: Define σ_m, define implementations, prove* •

Slogans and questions

- Algorithms compute functions from specific primitives
- They are specified by systems of recursive equations
- An algorithm is (faithfully modeled by) the semantic content of a system of recursive equations
- Machines are algorithms, but not all algorithms are machines
- Some algorithms have machine implementations
- An algorithm codes all its implementation-independent properties
- What is the relation between an algorithm and its implementations?
  ... or between two implementations of the same algorithm?

Main slogan

The theory of algorithms is the theory of recursive equations

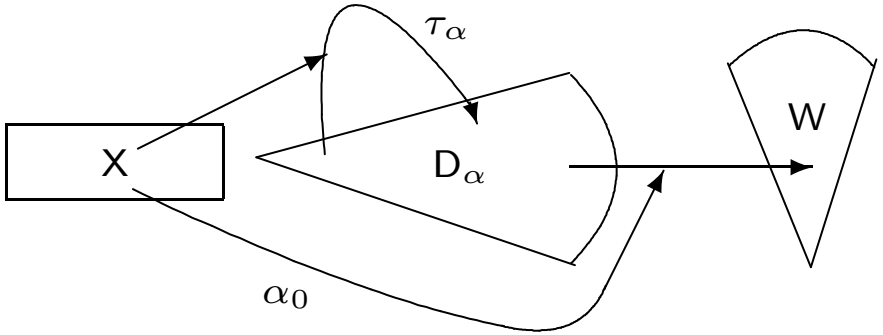
(Skip non-deterministic algorithms and fairness)

Monotone recursive equations

- A complete poset is a partial ordered set $D = (\mathrm{Field}(D), \leq_D)$ in which every directed set has a least upper bound
- Standard example:
  $(X \rightharpoonup Y) =$ the set of all partial functions $f : X \rightharpoonup Y$
- A function $f : D \to E$ is monotone if $x \leq_D y \implies f(x) \leq_E f(y)$
  ($f : X \rightharpoonup Y$ is a monotone function on X to $Y \cup \{\bot\}$)
- *For every monotone $f : D \to D$ on a complete D, the equation $\boxed{x = f(x)}$ has a least solution*

- Complete posets (domains) are the basic objects studied in Scott's Denotational Semantics for programming languages
- Much of this work can be viewed as a refinement of Denotational Semantics (which interprets programs by algorithms)

Recursors



A recursor $\alpha : X \rightsquigarrow W$ is a tuple $\alpha = (\alpha_0, \alpha_1, \ldots, \alpha_k)$ such that

1. X is a poset, W is a complete poset
2. $D_1, \ldots, D_k$ are complete posets, $D_\alpha = D_1 \times \cdots \times D_k$, the solution space of α
3. $\alpha_i : X \times D_\alpha \to D_i$ is monotone $(i = 1, \ldots, k)$
4. $\tau_\alpha(x, \vec{d}) = (\alpha_1(x, \vec{d}), \ldots, \alpha_k(x, \vec{d}))$ is the transition function, $\tau_\alpha : X \times D_\alpha \to D_\alpha$
5. $\alpha_0 : X \times D_1 \times \cdots \times D_k \to W$ is monotone, the output map

$\overline{\alpha}(x) = \alpha_0(x, \overline{d}_1, \ldots, \overline{d}_k)$ for the least solution of $\boxed{\vec{d} = \tau_\alpha(x, \vec{d})}$

We write $\quad \alpha(x) = \alpha_0(x, \vec{d})$ where $\{\vec{d} = \tau_\alpha(x, \vec{d})\}$

Recursor isomorphism

Two recursors

$$\alpha = (\alpha_0, \alpha_1, \ldots, \alpha_k), \quad \alpha' = (\alpha'_0, \alpha'_1, \ldots, \alpha'_m) : X \rightsquigarrow W$$

are isomorphic ($\alpha \simeq \alpha'$) if

(1) $k = m$ (same number of parts)

(2) There is a permutation $\pi : \{1, \ldots, k\}$ and poset isomorphisms $\rho_i : D_i \to D'_{\pi(i)}$ $(i = 1, \ldots, k)$ such that ...
(the order of the equations in the system $\vec{d} = \tau_\alpha(x, \vec{d})$ does not matter)

Isomorphic recursors $\alpha, \alpha' : X \rightsquigarrow W$ compute the same function $\overline{\alpha} = \overline{\alpha}' : X \to W$

Machines or recursors?

With each machine $\mathfrak{m} = (S, \text{input}, \sigma, T, \text{output}) : X \rightsquigarrow Y$ we associate the tail recursor

$$\mathfrak{r}_{\mathfrak{m}}(x) = p(\text{input}(x)) \text{ where}$$
$$\{p = \lambda(s)[\text{if } (s \in T) \text{ then output}(s) \text{ else } p(\sigma(s))]\}$$

- $\mathfrak{m}$ and $\mathfrak{r}_{\mathfrak{m}}$ compute the same partial function $\bar{\mathfrak{r}}_{\mathfrak{m}} = \overline{\mathfrak{m}} : X \rightharpoonup Y$
- **Theorem** (with V. Paschalis) The map $\mathfrak{m} \mapsto \mathfrak{r}_{\mathfrak{m}}$ respects isomorphisms, $\mathfrak{m} \simeq \mathfrak{m}' \iff \mathfrak{r}_{\mathfrak{m}} \simeq \mathfrak{r}_{\mathfrak{m}'}$
- The question is one of choice of terminology (because the mergesort system is also needed)
- Yuri Gurevich has argued that algorithms are machines (and of a very specific kind)
- Jean-Yves Girard has also given similar arguments

Elementary (first order) algorithms

Algorithms which compute partial functions from given partial functions

(Partial, pointed) algebra $\mathbf{M} = (M, 0, 1, \Phi^{\mathbf{M}})$

where $0, 1 \in M$, Φ is a set of function symbols (the vocabulary) and $\Phi^{\mathbf{M}} = \{\phi^{\mathbf{M}}\}_{\phi \in \Phi}$, with $\phi^{\mathbf{M}} : M^{n_\phi} \rightharpoonup M$ for each $\phi \in \Phi$

$\mathbf{N}_\varepsilon = (\mathbb{N}, 0, 1, \mathrm{rem})$, the Euclidean algebra
$\mathbf{N}_u = (\mathbb{N}, 0, 1, S, \mathrm{Pd})$, the *unary numbers*
$\mathbf{N}_b = (\mathbb{N}, 0, 1, \mathrm{Parity}, \mathrm{iq}_2, (x \mapsto 2x), (x \mapsto 2x + 1))$, the *binary numbers*
$\mathbf{A}^* = (A^*, 0, 1, \leq, \mathrm{head}, \mathrm{tail}, \ldots)$, the mergesort algebra, with $0, 1 \in A^*$

Standard model-theoretic notions must be mildly adapted, for example for (partial) subalgebras:

$$\mathbf{U} \subseteq_p \mathbf{M} \iff \{0, 1\} \subseteq U \subseteq M \text{ and for all } \phi, \phi^{\mathbf{U}} \subseteq \phi^{\mathbf{M}}$$

Recursive (McCarthy) programs of $\mathbf{M} = (M, 0, 1, \Phi^{\mathbf{M}})$

Explicit Φ-terms (with partial function variables and conditionals)

$$A :\equiv 0 \mid 1 \mid \mathrm{v}_i \mid \phi(A_1, \ldots, A_n) \mid \mathrm{p}_i^n(A_1, \ldots, A_n) \mid \text{if } (A = 0) \text{ then } B \text{ else } C$$

Recursive program (only $\vec{x}_i, p_1, \ldots, p_K$ occur in each part A_i):

$$A \quad : \quad \begin{cases} \mathrm{p}_A(\vec{\mathrm{x}}_0) = A_0 \\ \mathrm{p}_1(\vec{\mathrm{x}}_1) = A_1 \\ \vdots \\ \mathrm{p}_K(\vec{\mathrm{x}}_K) = A_K \end{cases} \quad (A_0 : \text{ the head}, (A_1, \ldots, A_K) : \text{ the body})$$

- What is the semantic content of the system A?

The recursor of a program in $\mathbf{M}$

$$A \quad : \quad \begin{cases} \mathrm{p}_A(\vec{\mathrm{x}}_0) = A_0 \\ \mathrm{p}_1(\vec{\mathrm{x}}_1) = A_1 \\ \vdots \\ \mathrm{p}_K(\vec{\mathrm{x}}_K) = A_K \end{cases}$$

$$\mathfrak{r}(A, \mathbf{M})(\vec{x}) = \mathrm{den}(A_0, \mathbf{M})(\vec{x}, \vec{p}) \text{ where}$$

$$\left\{ p_1 = \lambda(\vec{x}_1)\mathrm{den}(A_1, \mathbf{M})(\vec{x}_1, \vec{p}), \ldots, p_K = \lambda(\vec{x}_K)\mathrm{den}(A_K, \mathbf{M})(\vec{x}_K, \vec{p}) \right\}$$

$\mathfrak{r}(A, \mathbf{M})$ is not exactly the algorithm expressed by A in $\mathbf{M}$.

For example, if $\quad A \quad : \quad p_A(\vec{x}) = A_0(\vec{x})$ has empty body, then

$$\mathfrak{r}(A, \mathbf{M})(\vec{x}) = \mathrm{den}(A_0, \mathbf{M})(\vec{x}) \text{ where } \{ \quad \}$$

is just the function defined on $\mathbf{M}$ by A_0
(which may involve much explicit computation)

The problem of defining implementations

van Emde Boas:

> *Intuitively, a simulation of [one class of computation models] M by [another] M' is some construction which shows that everything a machine $M_i \in M$ can do on inputs x can be performed by some machine $M'_i \in M'$ on the same inputs as well;*

We will define a reducibility relation $\alpha \leq_r \beta$ and call a machine $\mathfrak{m}$ an implementation of α if $\alpha \leq_r \mathfrak{r}_{\mathfrak{m}}$

(where $\mathfrak{r}_{\mathfrak{m}}$ is the recursor representation of the machine $\mathfrak{m}$)

Recursor reducibility

Suppose $\alpha, \beta : X \rightsquigarrow W$, (e.g., $\beta = \mathfrak{r}_{\mathfrak{m}}$ where $\mathfrak{m} : X \rightsquigarrow W$):
A *reduction* of α to β is any monotone mapping

$$\pi : X \times D_\alpha \to D_\beta$$

such that the following three conditions hold, for every $x \in X$ and every $d \in D_\alpha$:

(R1) $\tau_\beta(x, \pi(x, d)) \leq \pi(x, \tau_\alpha(x, d))$.
(R2) $\beta_0(x, \pi(x, d)) \leq \alpha_0(x, d)$.
(R3) $\overline{\alpha}(x) = \overline{\beta}(x)$.

- $\alpha \leq_r \beta$ if a reduction exists
- $\mathfrak{m}$ implements α if $\alpha \leq_r \mathfrak{r}_{\mathfrak{m}}$

Implementations of elementary algorithms

Theorem (with Paschalis)

For any recursive program A in an algebra $\mathbf{M}$*, the standard implementation of A is an implementation of* $\mathfrak{r}(A, \mathbf{M})$

. . . *Uniformly* enough, so that (with the full definitions), the standard implementation of A implements the elementary algorithm expressed by A in $\mathbf{M}$

. . . And this is true of all familiar implementations of recursive programs

. . . so that the basic (complexity and resource use) upper and lower bounds established from the program A hold of all implementations of A

And for the applications to complexity theory, we work directly with the recursive equations of A

English as a programming language

Yiannis N. Moschovakis
UCLA and University of Athens

Tarski Lecture 2, March 5, 2008

Frege on sense

"*[the sense of a sign] may be the common property of many people*" Meanings are public (abstract?) objects

"*The sense of a proper name is grasped by everyone who is sufficiently familiar with the language . . . Comprehensive knowledge of the thing denoted . . . we never attain*"

Speakers of the language know the meanings of terms

"*The same sense has different expressions in different languages or even in the same language*"

"*The difference between a translation and the original text should properly not overstep the [level of the idea]*"

Faithful translation should preserve meaning

Outline of Lecture 2

Slogan:

The meaning of a term is the algorithm which computes its denotation

(1) Formal Fregean semantics in $\mathsf{L}_r^\lambda(K)$
(2) Meaning and synonymy in $\mathsf{L}_r^\lambda(K)$
(3) What are the objects of belief? (Local synonymy)
(4) The decision problem for synonymy

Sense and denotation as algorithm and value (1994)
A logical calculus of meaning and synonymy (2006)
Two aspects of situated meaning (with E. Kalyvianaki, to appear)
Posted in www.math.ucla.edu/∼ynm

The methodology of formal Fregean semantics

- An *interpreted formal language* L is selected
- The rendering operation on a fragment of English:

$$\text{English expression} + \text{informal context} \xrightarrow{\text{render}} \text{formal expression} + \text{state}$$

- Semantic values (denotations, meanings, etc.) are defined rigorously for the formal expressions of L and assigned to English expressions via the rendering operation

- Montague: L *should be a higher type language* (to interpret co-ordination, co-indexing, ...)
- Claim: L *should be a programming language* (to interpret self-reference and to define meanings properly)

The typed λ-calculus with recursion $\mathsf{L}_r^\lambda(K)$ - types

An extension of the typed λ-calculus, into which Montague's Language of Intensional Logic LIL can be easily interpreted (Gallin)

Basic types $b \equiv e \mid t \mid s$ (entities, truth values, states)

Types: $\sigma :\equiv b \mid (\sigma_1 \to \sigma_2)$

Abbreviation: $\sigma_1 \times \sigma_2 \to \tau \equiv (\sigma_1 \to (\sigma_2 \to \tau))$

Every non-basic type is uniquely of the form

$$\sigma \equiv \sigma_1 \times \cdots \times \sigma_n \to b$$

$\mathsf{level}(b) = 0$
$\mathsf{level}(\sigma_1 \times \cdots \times \sigma_n \to b) = \max\{\mathsf{level}(\sigma_1), \ldots, \mathsf{level}(\sigma_n)\} + 1$

The typed λ-calculus with recursion $\mathsf{L}_r^\lambda(K)$ - syntax

Pure variables: v_0^σ, v_1^σ, ..., for each type σ $(v : \sigma)$
Pure parameters: $\bar{u}$ for each state u (for convenience only)
Recursive variables: p_0^σ, p_1^σ, ..., for each type σ $(p : \sigma)$

Constants: A finite set K of typed constants (run, cow, he, the, every)

Terms – with assumed type restrictions and assigned types $(A : \sigma)$

$$A :\equiv v \mid \bar{u} \mid p \mid c \mid B(C) \mid \lambda(v)(B)$$
$$\mid A_0 \text{ where } \{p_1 = A_1, \ldots, p_n = A_n\}$$

$$C : \sigma, B : (\sigma \to \tau) \implies B(C) : \tau$$
$$v : \sigma, B : \tau \implies \lambda(v)(B) : (\sigma \to \tau)$$
$$A_0 : \sigma \implies A_0 \text{ where } \{p_1 = A_1, \ldots, p_n = A_n\} : \sigma$$

Abbreviation: $A(B, C, D) \equiv A(B)(C)(D)$

$\mathsf{L}_r^\lambda(K)$ - denotational semantics

- We are given basic sets $\mathbb{T}_s, \mathbb{T}_e$ and $\mathbb{T}_t \subseteq \mathbb{T}_e$ for the basic types

$$\mathbb{T}_{\sigma\to\tau} = \text{the set of all functions } f : \mathbb{T}_\sigma \to \mathbb{T}_\tau$$
$$\mathbb{P}_b = \mathbb{T}_b \cup \{\bot\} = \text{the "flat poset" of } \mathbb{T}_b$$
$$\mathbb{P}_{\sigma\to\tau} = \text{the set of all functions } f : \mathbb{T}_\sigma \to \mathbb{P}_\tau$$

$\mathbb{T}_\sigma \subseteq \mathbb{P}_\sigma$ and $\mathbb{P}_\sigma$ is a complete poset (with the pointwise ordering)

- We are given an object $c : \mathbb{P}_\sigma$ for each constant $\mathsf{c} : \sigma$

- Pure variables of type σ vary over $\mathbb{T}_\sigma$; recursive ones over $\mathbb{P}_\sigma$
- If $A : \sigma$ and π is a type-respecting assignment to the variables, then $\mathsf{den}(A)(\pi) \in \mathbb{P}_\sigma$
- Recursive terms are interpreted by the taking of least-fixed-points

Rendering natural language in $\mathsf{L}_r^\lambda(K)$

$$\tilde{t} \equiv (s \to t) \qquad \text{(type of Carnap intensions)}$$
$$\tilde{e} \equiv (s \to e) \qquad \text{(type of individual concepts)}$$

$$\text{Abelard loves Eloise} \xrightarrow{\text{render}} \text{loves(Abelard,Eloise)} : \tilde{t}$$
$$\text{Bush is the president} \xrightarrow{\text{render}} \text{eq(Bush,the(president))} : \tilde{t}$$
$$\text{liar} \xrightarrow{\text{render}} p \text{ where } \{p = \neg p\} : t$$
$$\text{truthteller} \xrightarrow{\text{render}} p \text{ where } \{p = p\} : t$$

$$\text{Abelard}, \text{Eloise}, \text{Bush} : \tilde{e}$$
$$\text{president} : \tilde{e} \to \tilde{t}, \text{eq} : \tilde{e} \times \tilde{e} \to \tilde{t}$$
$$\neg : t \to t, \text{the} : (\tilde{e} \to \tilde{t}) \to \tilde{e}$$

$$\text{den(liar)} = \text{den(truthteller)} = \bot$$

Co-ordination and co-indexing in $\mathsf{L}_r^\lambda(K)$

John stumbled and fell vs. John stumbled and he fell

John stumbled and fell $\xrightarrow{\text{render}}$ $\lambda(x)\Big(\text{stumbled}(x)\ \&\ \text{fell}(x)\Big)(\text{John})$

(predication after co-ordination)

This is in Montague's LIL (as it is interpreted in $\mathsf{L}_r^\lambda(K)$)

John stumbled and he fell $\xrightarrow{\text{render}}$ $\text{stumbled}(j)\ \&\ \text{fell}(j)$ where $\{j = \text{John}\}$

(conjunction after co-indexing)

The logical form of this sentence <u>cannot</u> be captured faithfully in LIL — recursion models co-indexing preserving logical form

Can we say nonsense in $\mathsf{L}_r^\lambda(K)$?

Yes!
In particular, we have parameters over states—so we can explicitly refer to the state (even to two states in one term); LIL does not allow this, *because we cannot do this in English*

Consider the terms

$$A \equiv \text{rapidly}(\text{tall})(\text{John}),\ B \equiv \text{rapidly}(\text{sleeping})(\text{John}) : \tilde{t}$$

A and B are terms of LIL,
not the renderings of correct English sentences

- The target formal language is a tool for defining rigorously the desired semantic values and it needs to be richer than a direct formalization of the relevant fragment of English
—to insure compositionality, if for no other reason

Meaning and synonymy in $\mathsf{L}_r^\lambda(K)$

- For a sentence $A : \tilde{t}$, the Montague sense of A is $\mathrm{den}(A) : \mathbb{T}_s \to \mathbb{T}_t$, so that

  there are infinitely many primes

  is Montague-synonymous with $1 + 1 = 2$

- In $\mathsf{L}_r^\lambda(K)$: *The meaning of a term A is modeled by an algorithm* $\mathrm{int}(A)$ *which computes* $\mathit{den}(A)(\pi)$ *for every* π
- The referential intension $\mathrm{int}(A)$ is compositionally determined from A
- $\mathrm{int}(A)$ is an abstract (not necessarily implementable) recursive algorithm of $\mathsf{L}_r^\lambda(K)$
- Referential synonymy: $\boxed{A \approx B \iff \mathrm{int}(A) \sim \mathrm{int}(A)}$

Reduction, Canonical Forms and the Synonymy Theorem

- A reduction relation $A \Rightarrow B$ is defined on terms of $\mathsf{L}_r^\lambda(K)$
- Each term A is reducible to a unique (up to congruence) irreducible recursive term, its canonical form

$$A \Rightarrow \mathrm{cf}(A) \equiv A_0 \text{ where } \{p_1 = A_1, \ldots, p_n = A_n\}$$

- $\boxed{\mathrm{int}(A) = (\mathrm{den}(A_0), \mathrm{den}(A_1), \ldots, \mathrm{den}(A_n))}$
- The parts $A_0, \ldots, A_n$ of A are irreducible, explicit terms
- $\mathrm{cf}(A)$ models the logical form of A
- *Synonymy Theorem. $A \approx B$ if and only if*

$$B \Rightarrow \mathrm{cf}(B) \equiv B_0 \text{ where } \{p_1 = B_1, \ldots, p_m = B_m\}$$

so that $n = m$ and for $i \leq n$, $den(A_i) = den(B_i)$

Is this notion of meaning Fregean?

Evans (in a discussion of Dummett's similar, computational interpretations of Frege's sense):

> *"This leads [Dummett] to think generally that the sense of an expression is (not a way of thinking about its [denotation], but) a method or procedure for determining its denotation. So someone who grasps the sense of a sentence will be possessed of some method for determining the sentence's truth value*
> *. . . ideal verificationism*
> *. . . there is scant evidence for attributing it to Frege"*

Converse question: For a sentence A, if you possess the method determined by A for determining its truth value, do you then "grasp" the sense of A?

(Sounds more like Davidson rather than Frege)

The reduction calculus

$$\text{Bush is the president} \xrightarrow{\text{render}} \text{eq}(\text{Bush})(\text{the}(\text{president}))$$

$$\Rightarrow \text{eq}(\text{Bush})(L) \text{ where } \{L = \text{the}(\text{president})\}$$

$$\Rightarrow \text{eq}(\text{Bush})(L) \text{ where } \{L = \text{the}(p) \text{ where } \{p = \text{president}\}\}$$

$$\Rightarrow \text{eq}(\text{Bush})(L) \text{ where } \{L = \text{the}(p), p = \text{president}\}$$

$$\Rightarrow \Big(\text{eq}(b) \text{ where } \{b = \text{Bush}\}\Big)(L) \text{ where } \{L = \text{the}(p), p = \text{president}\}$$

$$\Rightarrow \Big(\text{eq}(b)(L) \text{ where } \{b = \text{Bush}\}\Big) \text{ where } \{L = \text{the}(p), p = \text{president}\}$$

$$\Rightarrow_{cf} \boxed{\text{eq}(b)(L) \text{ where } \{b = \text{Bush}, L = \text{the}(p), p = \text{president}\}}$$

$$\text{He is the president} \xrightarrow{\text{render}} \text{eq}(\text{He})(\text{the}(\text{president}))$$

$$\Rightarrow_{cf} \boxed{\text{eq}(b)(L) \text{ where } \{b = \text{He}, L = \text{the}(p), p = \text{president}\}}$$

The reduction calculus

John loves and honors his father

$$\xrightarrow{\text{render}} \Big(\lambda(x)(\text{loves}(j,x)\ \&\ \text{honors}(j,x))\Big)(\text{father}(j)) \text{ where } \{j = \text{John}\}$$

$$\Rightarrow \Big[\Big(\lambda(x)(\text{loves}(j,x)\ \&\ \text{honors}(j,x))\Big)(f) \text{ where } \{f = \text{father}(j)\}\Big] \text{ where } \{j = \text{John}\}$$

$$\Rightarrow \Big(\lambda(x)(\text{loves}(j,x)\ \&\ \text{honors}(j,x))\Big)(f) \text{ where } \{f = \text{father}(j), j = \text{John}\}$$

$$\Rightarrow \Big(\lambda(x)\Big[(l\ \&\ h) \text{ where } \{l = \text{loves}(j,x), h = \text{honors}(j,x)\}\Big]\Big)(f) \text{ where } \{f = \text{father}(j), j = \text{John}\}$$

$$\Rightarrow \Big(\lambda(x)(l(x)\ \&\ h(x)) \text{ where } \{l = \lambda(x)\text{loves}(j,x), h = \lambda(x)\text{honors}(j,x)\}\Big)(f) \text{ where } \{f = \text{father}(j), j = \text{John}\}$$

$$\Rightarrow \lambda(x)(l(x)\ \&\ h(x))(f) \text{ where } \{l = \text{loves}(j,\cdot), h = \text{honors}(j,\cdot), f = \text{father}(j), j = \text{John}\}$$

Utterances, local meanings, local synonymy

An utterance is a pair (A, u), where A is a sentence, $A : \tilde{t}$ and u is a state; it is expressed in $\mathrm{L}_r^{\lambda}(K)$ by the term $A(\bar{u})$

The local meaning of A at the state u is $\mathrm{int}(A(\bar{u}))$

$$A \approx_u B \iff A(\bar{u}) \approx B(\bar{u})$$

Bush is the president$(\bar{u})$

$\Rightarrow_{cf} \mathrm{eq}(b)(L)(\bar{u})$ where $\{b = \mathrm{Bush}, L = \mathrm{the}(p), p = \mathrm{president}\}$

He is the president$(\bar{u})$

$\Rightarrow_{cf} \mathrm{eq}(b)(L)(\bar{u})$ where $\{b = \mathrm{He}, L = \mathrm{the}(p), p = \mathrm{president}\}$

Bush is the president $\not\approx_u$ He is the president

even if at the state $\bar{u}$, $\mathrm{He}(\bar{u}) = \mathrm{Bush}(\bar{u})$

Three aspects of meaning for a sentence $A : \tilde{t}$

Referential intension	$\mathrm{int}(A)$	Referential synonymy $\approx$
Local meaning at u	$\mathrm{int}(A(\bar{u}))$	Local synonymy $\approx_u$
Factual content at u	$\mathrm{FC}(A, u)$	Factual synonymy $\approx_{f,u}$

The *factual content* of a sentence at a state u gives a *representation of the world* at u (Eleni Kalyvianaki's Ph.D. Thesis)

Bush is the president $\not\approx_u$ He is the president

Bush is the president $\approx_{f,u}$ He is the president

Claim: *The objects of belief are local meanings*

The distinction between local meaning and factual content are related to David Kaplan's distinction between the *character* and *content* of a sentence at a state

Some referential (global) synonymies and non-synonymies

- There are infinitely many primes $\not\approx 1 + 1 = 2$
- $A \;\&\; B \approx B \;\&\; A$
- The morning star is the evening star
  $\approx$The evening star is the morning star
  (This fails with Montague's renderings)
- Abelard loves Eloise $\approx$ Eloise $\boxed{\text{is loved by}}$ Abelard (Frege)
- $2 + 3 = 6 \approx 3 + 2 = 6$ (with $+$ and the numbers primitive)
- liar $\not\approx$ truthteller
- John stumbled and he fell $\xrightarrow{\text{render}}$
  $A \equiv \text{stumbled}(j) \;\&\; \text{fell}(j)$ where $\{j = \text{John}\}$
  A is not $\approx$ with any *explicit* term (including any term from LIL)

Is referential synonymy decidable?

Synonymy Theorem. $A \approx B$ if and only if

$$A \Rightarrow \mathrm{cf}(A) \equiv A_0 \text{ where } \{p_1 = A_1, \ldots, p_n = A_n\}$$
$$B \Rightarrow \mathrm{cf}(B) \equiv B_0 \text{ where } \{p_1 = B_1, \ldots, p_n = B_n\}$$

so that for $i = 0, \ldots, n$ and all π, $den(A_i)(\pi) = den(B_i)(\pi)$.

- Synonymy is reduced to denotational equality for explicit, irreducible terms (the truth facts of A)
- Denotational equality for arbitrary terms is undecidable (there are constants, with fixed interpretations)
- The explicit, irreducible terms are very special
  — but by no means trivial!

The synonymy problem for $\mathsf{L}^{\lambda}_{r}(K)$ (with finite K)

- The decision problem for $\mathsf{L}^{\lambda}_{r}(K)$-synonymy is open

Theorem *If the set of constants K is finite, then synonymy is decidable for terms of* **adjusted level** ≤ 2

These include terms constructed "simply" from

Names of "pure" objects	$0, 1, 2, \emptyset, \ldots$	$: e$
Names, demonstratives	John, I, he, him	$: \tilde{e}$
Common nouns	man, unicorn, temperature	$: \tilde{e} \to \tilde{t}$
Adjectives	tall, young	$: (\tilde{e} \to \tilde{t}) \to (\tilde{e} \to \tilde{t})$
Propositions	it rains	$: \tilde{t}$
Intransitive verbs	stand, run, rise	$: \tilde{e} \to \tilde{t}$
Transitive verbs	find, loves, be	$: \tilde{e} \times \tilde{e} \to \tilde{t}$
Adverbs	rapidly	$: (\tilde{e} \to \tilde{t}) \to (\tilde{e} \to \tilde{t})$

Proof is by reducing this claim to the Main Theorem in the 1994 paper (for a corrected version see www.math.ucla.edu/∼ynm)

Explicit, irreducible identities that must be known

- Los Angeles = LA (Athens = Αθήνα)
- $x \;\&\; y = y \;\&\; x$
- $\text{between}(x, y, z) = \text{between}(x, z, y)$
- $\text{love}(x, y) = \text{be\_loved}(y, x)$

A dictionary is needed—but what kind and how large?

$$\text{ev}_2(\lambda(u_1, u_2)r(u_1, u_2, \vec{a}), b, z) = \text{ev}_1(\lambda(v)r(v, z, \vec{a}), b)$$

Evaluation functions: both sides are equal to $r(b, z, \vec{a})$

The dictionary line which determines this is (essentially)

$$\lambda(s)x(s, z) = \lambda(s)y(s) \implies \text{ev}_2(x, b, z) = \text{ev}_1(y, b)$$

The form of the decision algorithm

- A <u>finite</u> list of true dictionary lines is constructed, which codifies the relationships between the constants
- Given two explicit, irreducible terms A, B of adjusted level ≤ 2, we construct (effectively) a finite set $L(A, B)$ of lines such that

$$\begin{aligned} &\models A = B \\ \iff &\text{ every line in } L(A, B) \text{ is congruent to one in the dictionary} \end{aligned}$$

- It is a lookup algorithm, justified by a finite basis theorem
- Complexity: NP; the graph isomorphism problem is reducible to the synonymy problem for very simple (propositional) recursive terms

The axiomatic derivation of absolute lower bounds

Yiannis N. Moschovakis
UCLA and University of Athens

Tarski Lecture 3, March 7, 2008

A lower bound result

Theorem (van den Dries, ynm)

If an algorithm α decides the coprimeness relation $x \perp\!\!\!\perp y$ on $\mathbb{N}$ from the primitives $\leq, +, \dot{-}, \mathrm{iq}, \mathrm{rem}$, then for infinitely many a, b

$$c^s_\alpha(a,b) > \frac{1}{10}\log\log(\max(a,b)) \qquad (*)$$

In fact (*) holds for all solutions of Pell's equation, $a^2 = 1 + 2b^2$

- $\mathrm{iq}(x,y), \mathrm{rem}(x,y)$ are the integer quotient and remainder
- $c^s_\alpha(x,y)$ counts the number of applications of the primitives in the computation
- Claim: This applies to all algorithms from the specified primitives
- The Euclidean decides coprimeness from rem with complexity

$$c^s_\epsilon(a,b) \leq 2\log(\min(a,b)) \quad (\min(a,b) \geq 2)$$

Outline of Lecture 3

Slogan: *Lower bound results*
are the undecidability facts about decidable problems

. . . and so they should be (to some extent) a matter of logic

(1) Tweak logic (a bit) so it applies smoothly to computation theory
(2) Three (simple) axioms for elementary algorithms,
a la *abstract model theory*
(3) Lower bounds from the axioms
(4) Lower bounds for elementary algorithms on logical extensions

Is the Euclidean algorithm optimal among its peers? (with vDD, 2004)
Arithmetic complexity (with vDD, to appear)

Partial algebras, embeddings and subalgebras

- A (Partial, pointed) algebra is a structure $\mathbf{M} = (M, 0, 1, \Phi^{\mathbf{M}})$ where $0, 1 \in M$, Φ is a set of function symbols (the vocabulary) and $\Phi^{\mathbf{M}} = \{\phi^{\mathbf{M}}\}_{\phi \in \Phi}$, with $\phi^{\mathbf{M}} : M^{n_\phi} \rightharpoonup M$ for each $\phi \in \Phi$

- An embedding $\iota : \mathbf{U} \rightarrowtail \mathbf{M}$ from one Φ-algebra into another is any injection $\iota : U \rightarrowtail M$ such that

$$\iota(0^{\mathbf{U}}) = 0^{\mathbf{M}}, \quad \iota(1^{\mathbf{U}}) = 1^{\mathbf{M}},$$

  and for all $\phi \in \Phi, x_1, \ldots, x_n, w \in U$,

$$\phi^{\mathbf{U}}(x_1, \ldots, x_n) = w \implies \phi^{\mathbf{M}}(\iota x_1, \ldots, \iota x_n) = \iota w$$

- $\mathbf{U} \subseteq_p \mathbf{M}$ if the identity $I : U \rightarrowtail M$ is an embedding

Algebra restrictions

$\mathbf{N}_\varepsilon = (\mathbb{N}, 0, 1, \mathrm{rem})$, the Euclidean algebra
$\mathbf{N}_u = (\mathbb{N}, 0, 1, S, \mathrm{Pd})$, the *unary numbers*
$\mathbf{N}_b = (\mathbb{N}, 0, 1, \mathrm{Parity}, \mathrm{iq}_2, (x \mapsto 2x), (x \mapsto 2x+1))$, the *binary numbers*

For $\mathbf{M} = (M, 0, 1, \Phi^{\mathbf{M}})$ and $\{0, 1\} \subseteq U \subseteq M$, let

$$\mathbf{M} \restriction U = (U, 0, 1, \Phi^{\mathbf{U}}),$$

where for $\phi \in \Phi$,

$$\phi^{\mathbf{U}}(\vec{x}) = w \iff \vec{x}, w \in U \ \& \ \phi^{\mathbf{M}}(\vec{x}) = w$$

For finite $U \subset \mathbb{N}$, $\mathbf{N}_u \restriction U$ is a finite, properly partial subalgebra of $\mathbf{N}$

Subalgebras generated from the input, $\mathbf{M} = (M, 0, 1, \Phi^{\mathbf{M}})$

For $\vec{x} = x_1, \ldots, x_n \in M$, set

$$G_0(\vec{x}) = \{0, 1, x_1, \ldots, x_n\}$$
$$G_{m+1}(\vec{x}) = G_m(\vec{x}) \cup \{\phi^{\mathbf{M}}(\vec{u}) \mid \phi \in \Phi, \vec{u} \in G_m(\vec{x}) \text{ and } \phi^{\mathbf{M}}(\vec{u})\downarrow\}$$

so that

$$G_m(\vec{x}) = \{t^{\mathbf{M}}[x_1, \ldots, x_n] \in M \mid t(v_1, \ldots, v_n) \text{ is a term of depth} \leq m\}$$

$$\boxed{\mathbf{M} \restriction G_m(\vec{x}) \text{ is the subalgebra of depth } m \text{ generated by } \vec{x}}$$

$(\mathbf{M} \restriction \bigcup_m G_m(\vec{x}) \text{ is the subalgebra generated by } \vec{x})$

I The Locality Axiom

An algorithm α of arity n of an algebra $\mathbf{M} = (M, 0, 1, \Phi^{\mathbf{M}})$ assigns to each subalgebra $\mathbf{U} \subseteq_p \mathbf{M}$ an n-ary, strict partial function

$$\overline{\alpha}^{\mathbf{U}} : U^n \rightharpoonup U$$

- $\mathbf{M}$-algorithms "compute" strict partial functions, and they can be localized (relativized) to arbitrary subalgebras of $\mathbf{M}$

We write

$$\boxed{\mathbf{U} \models \overline{\alpha}(\vec{x}) = w \iff \vec{x} \in U^n, w \in U \text{ and } \overline{\alpha}^{\mathbf{U}}(\vec{x}) = w}$$

II The Embedding Axiom

If α is an n-ary algorithm of $\mathbf{M}$, $\mathbf{U}, \mathbf{V} \subseteq_p \mathbf{M}$, *and* $\iota : \mathbf{U} \rightarrowtail \mathbf{V}$ *is an embedding, then*

$$\mathbf{U} \models \overline{\alpha}(\vec{x}) = w \implies \mathbf{V} \models \overline{\alpha}(\iota\vec{x}) = \iota w \quad (x_1, \ldots, x_n, w \in U)$$

In particular, if $\mathbf{U} \subseteq_p \mathbf{M}$, then $\overline{\alpha}^{\mathbf{U}} \subseteq \overline{\alpha}^{\mathbf{M}}$

- An algorithm treats the primitives of $\mathbf{M}$ as *oracles*: it can request values $\phi^{\mathbf{M}}(\vec{y})$, and use them if they are provided

III The Finiteness Axiom

If α is an n-ary algorithm of $\mathbf{M}$*, then*

$$\mathbf{M} \models \overline{\alpha}(\vec{x}) = w \implies \textit{there is an } m \textit{ such that } \vec{x}, w \in G_m(\vec{x}) \text{ and } \mathbf{M} \restriction G_m(\vec{x}) \models \overline{\alpha}(\vec{x}) = w$$

In particular,

$$\overline{\alpha}^{\mathbf{M}}(\vec{x})\downarrow \implies \overline{\alpha}(\vec{x}) \in \bigcup_m G_m(\vec{x})$$

- "The computation" of $\overline{\alpha}^{\mathbf{M}}(\vec{x})$ takes place within the subalgebra of $\mathbf{M}$ generated by the input, and it is finite: take m large enough so that every y used in "the computation" is in $G_m(\vec{x})$

All algorithms—really—satisfy these axioms

- Explicit computation: $\overline{\alpha}^{\mathbf{U}}(\vec{x}) = t^{\mathbf{U}}[\vec{x}]$, where $t(\vec{v})$ is a Φ-term
- $\overline{\alpha}^{\mathbf{U}}$ is the partial function computed a fixed recursive (McCarthy) program A in the signature Φ (as in Lecture 1)
- $\overline{\alpha}^{\mathbf{U}}$ is computed by a register machine (or RAM, or Turing machine or ...) from $\Phi^{\mathbf{U}}$
- $\overline{\alpha}^{\mathbf{U}}$ is computed in Plotkin's PCF above the algebra $\mathbf{U}$
- $\overline{\alpha}^{\mathbf{U}}$ by computed in non-deterministic versions of any of these

Axioms for elementary algorithms

- I, Locality Axiom: An algorithm α of arity n of an algebra $\mathbf{M} = (M, 0, 1, \Phi^{\mathbf{M}})$ assigns to each subalgebra $\mathbf{U} \subseteq_p \mathbf{M}$ an n-ary, strict partial function

$$\overline{\alpha}^{\mathbf{U}} : U^n \rightharpoonup U \qquad (\mathbf{U} \models \overline{\alpha}(\vec{x}) = w \iff \overline{\alpha}^{\mathbf{U}}(\vec{x}) = w)$$

- II, Embedding Axiom: If $\mathbf{U}, \mathbf{V} \subseteq_p \mathbf{M}$, and $\iota : \mathbf{U} \rightarrowtail \mathbf{V}$ is an embedding, then

$$\mathbf{U} \models \overline{\alpha}(\vec{x}) = w \implies \mathbf{V} \models \overline{\alpha}(\iota\vec{x}) = \iota w \quad (x_1, \ldots, x_n, w \in U)$$

- III, Finiteness Axiom:

$$\mathbf{M} \models \overline{\alpha}(\vec{x}) = w \implies \textit{there is an } m \textit{ such that } \vec{x}, w \in G_m(\vec{x}) \\ \text{and } \mathbf{M} \restriction G_m(\vec{x}) \models \overline{\alpha}(\vec{x}) = w$$

The embedding complexity of an algorithm

If α is an algorithm of $\mathbf{M}$ and $\mathbf{M} \models \overline{\alpha}(\vec{x}) = w$, set

$$\boxed{c^{\iota}_{\alpha}(\vec{x}) = \text{the least } m \text{ such that } \mathbf{M} \restriction G_m(\vec{x}) \models \overline{\alpha}(\vec{x}) = w}$$

This is defined by the Finiteness Axiom

- Intuitively, if $m = c^{\iota}_{\alpha}(\vec{x})$, then any implementation of α will need to "consider" (use) some $u \in M$ of depth m; and so it will need at least m steps to construct this u from the input using the primitives
- If $\overline{\alpha}(\vec{x}) = t^{\mathbf{M}}[\vec{x}]$, then $c^{\iota}_{\alpha}(\vec{x}) \leq \text{depth}(t(\vec{v}))$
- c^{ι}_{α} is majorized by all usual time-complexity measures, including the number of calls to the primitives

The embedding complexity of a (computable) function

Fix $f : M^n \to M$. An embedding $\iota : \mathbf{M} \restriction G_m(\vec{x}) \rightarrowtail \mathbf{M}$ respects f at $\vec{x}$ if

$$f(\vec{x}) \in G_m(\vec{x}) \ \& \ \iota(f(\vec{x})) = f(\iota(\vec{x}))$$

Lemma

If some algorithm computes f in $\mathbf{M}$, then for each $\vec{x}$, there is some m such that every embedding $\iota : \mathbf{M} \restriction G_m(\vec{x}) \rightarrowtail \mathbf{M}$ respects f at $\vec{x}$

Proof Take $m = c^{\iota}_{\alpha}(\vec{x})$ for some α such that $f = \overline{\alpha}^{\mathbf{M}}$

$$\boxed{c^{\iota}_{f}(\vec{x}) = \text{the least } m \text{ such that every } \iota : \mathbf{M} \restriction G_m(\vec{x}) \rightarrowtail \mathbf{M} \text{ respects } f \text{ at } \vec{x}}$$

If α computes f in $\mathbf{M}$, then $c^{\iota}_{f}(\vec{x}) \leq c^{\iota}_{\alpha}(\vec{x})$

- To show that m is an absolute lower bound for the computation of $f(\vec{x})$ show that $\boxed{f(\vec{x}) \notin G_m(\vec{x})}$,
  or construct $\iota : \mathbf{M} \restriction G_m(\vec{x}) \rightarrowtail \mathbf{M}$ such that $\boxed{\iota f(\vec{x}) \neq f(\iota \vec{x})}$

Outline of a proof

Theorem (van den Dries, ynm)

For the algebra $\mathbf{M} = (\mathbb{N}, 0, 1, \leq, +, \dot{-}, \mathrm{iq}, \mathrm{rem})$ *and the relation of coprimeness* $x \perp\!\!\!\perp y$,

$$a^2 = 1 + 2b^2 \implies c^{\iota}_{\perp\!\!\!\perp}(a, b) > \frac{1}{10} \log\log(a) \qquad (*)$$

So if α *decides coprimeness in* $\mathbf{M}$, *then* (*) *holds with* $c^{\iota}_{\alpha}(a, b)$

- If $2^{2^{4m+6}} \leq a$, then every $X \in G_m(a, b)$ can be written uniquely as

$$X = \frac{x_0 + x_1 a + x_2 b}{x_3} \qquad \text{with } x_i \in \mathbb{Z}, \quad |x_i| \leq 2^{2^{4m}}$$

  and we can define $\iota : \mathbf{M} \restriction G_m(a, b) \rightarrowtail \mathbf{M}$ using $\lambda = 1 + a!$,

$$\iota(X) = \frac{x_0 + x_1 \lambda a + x_2 \lambda b}{x_3}, \text{ so } (\iota(a), \iota(b)) = (\lambda a, \lambda b)$$

$\mathbf{M} = (\mathbb{N}, 0, 1, \text{Parity}, \text{iq}_2, \leq, +, \dot{-}, \text{Presburger functions})$

- (van den Dries, ynm) *If $R(x)$ is one of the relations*

  $$x \text{ is prime}, \quad x \text{ is a perfect square}, \quad x \text{ is square free},$$

  then for some $r > 0$ and infinitely many a, $c^{\iota}_{R}(a) > r \log(a)$

- (van den Dries, ynm) *For some $r > 0$ and infinitely many a, b,*

  $$c^{\iota}_{\perp\!\!\!\perp}(a, b) > r \log(\max(a, b))$$

- (Joe Busch) *If $R(x, p) \iff x$ is a square* mod p,
  then for some $r > 0$ and a sequence (a_n, p_n) with $p_n \to \infty$,

  $$c^{\iota}_{R}(a_n, p_n) > r \log(p_n)$$

In the last two examples, the results match up to a multiplicative constant the known binary algorithms, so these are optimal

Primality in $\mathbf{M} = (\mathbb{N}, 0, 1, \mathrm{Parity}, \mathrm{iq}_2, \leq, +, \dot{-}, \mathrm{Presburger})$

Theorem (van den Dries, ynm)

If $\mathrm{Prime}(p) \iff p$ is prime, *then in* $\mathbf{M}$, *for some* $r > 0$ *and all primes* p,

$$c^{\iota}_{\mathrm{Prime}}(p) > r \log p \qquad (*)$$

So if α *decides primality in* $\mathbf{M}$, *then* (*) *holds with* $c^{\iota}_{\alpha}(p)$

- If $2^{2m+2} \leq a$, then every $X \in G_m(a)$ can be written uniquely as

$$X = \frac{x_0 + x_1 a}{2^m} \quad \text{with } |x_i| \leq 2^{2m},$$

  and we can define $\iota : \mathbf{M} \restriction G_m(a) \rightarrowtail \mathbf{M}$ by

$$\iota(X) = \frac{x_0 + x_1 \lambda a}{2^m}, \text{ with } \lambda = 1 + 2^m, \text{ so } \iota(a) = \lambda a$$

Primality in binary

- If $\mathrm{Prime}(p) \iff p$ is prime, then in

$$\mathbf{N}_b = (\mathbb{N}, 0, 1, \mathrm{Parity}, \mathrm{iq}_2, (x \mapsto 2x), (x \mapsto 2x+1))$$

  for some $r > 0$ and all primes p,

$$c^{\iota}_{\mathrm{Prime}}(p) \geq r \log p \qquad (*)$$

- This should follow trivially from number-theoretic results, because it takes at least i applications of the primitives of $\mathbf{N}_b$ to read i bits of the input; we should have $r = 1$
- Theorem (Tao). *For infinitely many primes p, if p' is constructed by changing any bit in the binary expansion of p except the highest, then p' is not prime*
- Tao found subsequently that this result is implicit in a paper of Cohen and Selfridge from 1975 and explicitly noted in a 2000 paper by Sun, and he obtained more general results

Non-uniform complexity

What if you are only interested in deciding $R(\vec{x})$ for n-bit numbers ($< 2^n$) and you are willing to use a different algorithm for each n?

Theorem (The lookup algorithm)

For each k-ary relation R on $\mathbb{N}$ and each n, there is an $\mathbf{N}_b$-term (with conditionals) $t_n(\vec{v})$ *of depth* $\leq n = \log_2(2^n)$ *which decides* $R(\vec{x})$ *for all* $\vec{x} < 2^n$.

- Non-uniform lower bounds are never greater than log
- The best ones establish the optimality of the lookup algorithm (and are most interesting when some uniform algorithm matches the lookup up to a multiplicative constant)
- They are mostly about "the size" of $t(\vec{v})$
- They do not follow from Axiom I – III

Recursive (McCarthy) programs of $\mathbf{M} = (M, 0, 1, \Phi^{\mathbf{M}})$

Explicit Φ-terms (with p_i^n partial function variables)

$$A :\equiv 0 \mid 1 \mid \mathrm{v}_i \mid \phi(A_1, \ldots, A_n) \mid \mathrm{p}_i^n(A_1, \ldots, A_n) \mid \text{if } (A_0 = 0) \text{ then } A_1 \text{ else } A_2,$$

Recursive program (only $\vec{x}_i, p_1, \ldots, p_K$ occur in each part A_i):

$$A \quad : \quad \begin{cases} \mathrm{p}_A(\vec{\mathrm{x}}_0) = A_0 \\ \mathrm{p}_1(\vec{\mathrm{x}}_1) = A_1 \\ \qquad\vdots \\ \mathrm{p}_K(\vec{\mathrm{x}}_K) = A_K \end{cases} \quad (A_0 : \text{ the head}, (A_1, \ldots, A_K) : \text{ the body})$$

The elementary algorithms of $\mathbf{M}$ *are expressed by recursive programs*

(and they satisfy Axioms I – III)

A non-uniform lower bound result for elementary algorithms

If α is the algorithm expressed by a recursive program in **M**, let

$$c^s_\alpha(\vec{x}) = \text{the number of calls to the primitives made in the computation of } \overline{\alpha}(\vec{x}) \ \geq c^\iota_\alpha(\vec{x})$$

Theorem (van den Dries, ynm)

Let $\mathbf{M} = (\mathbb{N}, 0, 1, \leq, +, \dot{-}, \mathrm{iq}, \mathrm{rem})$. *There is some* $r > 0$, *such that for all sufficiently large* n *and every* **M**-*elementary algorithm* α *which decides coprimeness for all* $x, y < 2^n$, *there exist* $a, b < 2^n$ *such that*

$$c^s_\alpha(a, b) > r \log_2 n \geq r \log_2 \log_2(\max(a, b))$$

The proof is by the embedding method, but uses special properties of recursive programs (the computation space)

Logical extensions (a la Tarski)

A $(\Phi \cup \Psi)$-algebra $\overline{\mathbf{M}}$ is a logical extension of a Φ-algebra $\mathbf{M}$ if

(1) $M \subseteq \overline{M}$, $0^{\mathbf{M}} = 0^{\overline{\mathbf{M}}}$, $1^{\mathbf{M}} = 1^{\overline{\mathbf{M}}}$

(2) For each $\phi \in \Phi$, $\phi^{\mathbf{M}} = \phi^{\overline{\mathbf{M}}}$

(3) Every bijection $\iota : M \rightarrowtail\!\!\!\!\twoheadrightarrow M$ which fixes $0, 1$ can be extended to a bijection $\bar{\iota} : \overline{M} \rightarrowtail\!\!\!\!\twoheadrightarrow \overline{M}$ such that for every $\psi \in \Psi$,

$$\psi^{\overline{\mathbf{M}}}(\bar{\iota}\vec{x}) = \bar{\iota}\psi^{\overline{\mathbf{M}}}(\vec{x}) \qquad (\vec{x} \in \overline{M}^n)$$

i.e., $\bar{\iota}$ is an automorphism of the reduct $(\overline{M}, 0, 1, \Psi^{\overline{\mathbf{M}}})$

- Random Access (and all other kinds of) Machines from $\Phi^{\mathbf{M}}$, Plotkin's PCF over $\mathbf{M}$, etc., are all faithfully represented by recursive programs on logical extensions of $\mathbf{M}$

The persistence of embedding complexity

Theorem (van den Dries, Neeman, ynm)

If $f : M^n \to M$ and $\overline{\mathbf{M}}$ is a logical extension of $\mathbf{M}$, then

$$c_f^\iota(\vec{x}, \mathbf{M}) = c_f^\iota(\vec{x}, \overline{\mathbf{M}}) \quad (\vec{x} \in M^n)$$

- This is why the embedding method gives the same lower bounds (for a function f from specified primitives) for RAMs and for recursive programs, even though the direct *simulation* of RAMs by recursive programs has an overhead
- The basic non-uniform results obtained by the embedding method also extend to arbitrary logical extensions

Back to sorting

Theorem

If $\leq$ is an ordering of a set A, $\overline{\mathbf{A}}$ is a logical extension of $(A \cup \{0,1\}, 0, 1, \leq)$ such that $A^* \subseteq A$, and α is an elementary algorithm of $\overline{\mathbf{A}}$ which sorts the sequences in A^*, then

$$|u| = n \implies c^s_\alpha(u) \geq \log_2(n!) \sim n \log_2(n),$$

where $c^s_\alpha(u)$ is the number of comparisons made by α in the computation of $\mathrm{sort}(u)$

This is proved by the classical, counting argument