ABOUT AND AROUND COMPUTING OVER THE REALS

Solomon Feferman

Logic Seminar, Stanford, April 17, 2012

Two Competing Theories of Computing over R

- Two competing theories of computing over the reals:
- The BSS (Blum-Shub-Smale) model
- The “bit” computation model (Banach-Mazur-Grzegorczyck)
- Each [recently] claims to be the proper foundation of scientific computing and computational complexity

The BSS model

- Full exposition of the BSS model and applications in Blum, Cuker, Shub, Smale, *Complexity and Real Computation* (1997)
- Nice exposition in Lenore Blum, "Computing over the reals: Where Turing meets Newton", *Notices AMS* 2004

The "Bit" Computation Model

- The "bit" computation (or "effective approximation") model: Banach and Mazur ideas, 1930s; developed by Grzegorczyck and, independently, Lacombe, 1955.
- Nice exposition by Mark Braverman and Stephen Cook in "Computing over the reals: Foundations for scientific computation", *Notices AMS* 2006.

These Theories are Incompatible

- Examples of incompatibility:
- The exponential function is computable in the effective approx. model but not in the BSS model.
- Given a polynomial $p(x)$ over Q, the function $f(x)=1$ if $p(x) = 0$, else 0, is computable in the BSS model but not (in general) in the effective approximation model.

Can Both be Reasonable?

- The BSS model is a reasonable theory of computation over R as an *algebraic structure*.
- The eff. approx. model is a reasonable theory of computation over R as a *topological structure* or as a *second-order structure*.

Subsuming Both Under Generalized Recursion Theories (g.r.t.)

- Turing and Register computability over arbitrary algebraic structures (Friedman 1971)
- "While" computation schemata over arb. algebraic and topological structures (Tucker and Zucker 2000)
- Higher type LFP schemata over arb. structures (Platek 1966, Moschovakis 1989, Feferman 1992)

The BSS model

- The BSS model makes sense over any ring A, possibly ordered.
- A BSS algorithmic procedure is given by a directed graph; top node for inputs, successor node for polynomial computation node, branching node on test for = (or <). Also described in terms of generalized Turing machines or register machines.
- Finite-dimensional case uses sequences of fixed length, infinite dim. case sequences of arb. length.

Examples of BSS Algorithms

- Newton algorithm for R or C. Given a rational fn. f and $\varepsilon > 0$, find a zero of f within accuracy ε: start with an input x, update by $x \to (x - f(x)/f'(x))$ until reach $|f(x)| < \varepsilon$. [A finite dim. case]
- Hilbert's Nullstellensatz. Given m polynomials in n variables over R or C, decide whether or not they have a common zero. [An infinite dim. case]

The BSS Model and Complexity

- The Mandelbrot set is not BSS-computable.
- Its complement is semi-computable; that can be used to “draw” it.
- Notions of P/A and NP/A for any ring A.
- Transfer Thm. P/C = NP/C iff P/A = NP/A for any alg. closed field A of char. 0.

The Effective Approximation Model[s]

- Explain for R, but generalizes to any complete separable metric space.
- Sequential (S-) effective approximation and Polynomial (P-) effective approximation.

S-Approximation Computability

- Let I be a finite or infinite interval in R.
- In order to define $f : I \to R$ effectively, find a computable functional F which, given x in I, maps any Cauchy sequence s of rationals approaching x to F(s), a Cauchy sequence approaching f(x).

S-Approximation Computability (cont'd)

- For simplicity, use approximations to reals x by sequences of dyadic rationals $\varphi(n)/2^n$ where $\varphi: N \to Z$ and
- (i) $|x - \varphi(n)/2^n| \leq 1/2^n$ for all n.
- Then find computable $F : (N \to Z) \to (N \to Z)$ such that whenever (i) holds and $F(\varphi) = \psi$ then
- (ii) $|f(x) - \psi(m)/2^m| \leq 1/2^m$ for all m.

The S-Eff. Approx. Functionals

- Using the effective correspondence of Z with N, this reduces to telling which functionals $F:(N \to N) \to (N \to N)$ are effectively computable.
- Let $\mathcal{T}$ be the class of total φ from N to N and $\mathcal{P}$ the class of partial φ from N to N.

The S-Eff. Approx. Functionals (con’td)

- Define: F from $\mathcal{T}$ to $\mathcal{T}$ is eff. computable iff it is the restriction to $\mathcal{T}$ of a partial recursive functional Φ from $\mathcal{P}$ to $\mathcal{P}$.
- Alternative characterization (Grzegorczyck): F is eff. computable if it is generated by the primitive recursive and μ (min operator) schemata for functionals on $\mathcal{T}$ to $\mathcal{T}$. [Analogous to Kleene’s schemata for general rec. fns.]
- Cf. also Weirauch (2000) TTE uniform oracle computability.

Continuity and P-Eff. Approx. Functions

- Theorem. If $f : I \to R$ is S-Approx. effectively computable, then f is continuous on I.
- Weierstrass Approximation Theorem. Each continuous f on a closed interval I is uniformly approximable by polynomials over Q.
- P-Approximation theory (Pour-El 1974): Use (effective) sequences of polynomials over Q to directly approximate (computable) f.

Complexity in S-approx. Theory

- P, NP etc. defined for S-approx. functions and functionals in Ker-I Ko (1991). ("In P, or not in P, that is the question.")
- Differentiation does not preserve P-time.
- Integration of f is P-time for all P-time computable f iff there is a collapse in a certain hierarchy.

Relevance to Scientific Computation?

- Scientific computation (aka numerical analysis): techniques for solving one or more linear or polynomial eqns., interpolation, numerical integration and differentiation, max and mins, optimization, numerical soln. of differential and integral eqns. , etc.
- Classic algorithms: Newton method, Lagrange interpolation, Gaussian elimination, Euler's method, etc. Modern use of computers.
- Uses "floating point arithmetic," error estimates.

The View From GRT: Register Machines on 1st OrderStructures

- Register machine computability on arbitrary first-order (possibly) many-sorted structures $\mathfrak{A}$ (Friedman 1971).
- $\mathfrak{A}$ may have one or more basic domains, operations on those domains, relations between those domains and designated constants. Equality on a given domain may or may not be included among the basic relations.

Register Machine Procedures

- "Finite algorithmic procedures" (fap)
- Given $\mathfrak{A}$, (i) enter inputs from $\mathfrak{A}$; (ii) set a register to a constant from $\mathfrak{A}$; (iii) perform one of the $\mathfrak{A}$-operations on register contents; (iv) test for one of the $\mathfrak{A}$-relations on register contents and branch according to instructions.
- FAP(A) = the partial fap computable fns.

Extensions of FAP Computability

- Let $\mathfrak{N}$ = (N, Sc, Pd, 0, =). Then FAP($\mathfrak{N}$) is the set of all partial recursive functions.
- Define FAPC($\mathfrak{A}$) = FAP($\mathfrak{A}$, $\mathfrak{N}$), "faps with counting".
- Take $\mathfrak{A}^*$ to be given by arbitrary finite sequences (or "stacks") for each domain of A, with operations of adding ("push") and deleting at the end ("pop").
- Define FAPS($\mathfrak{A}$) = FAP($\mathfrak{A}$, $\mathfrak{A}^*$) and FAPCS($\mathfrak{A}$) = FAP($\mathfrak{A}$, $\mathfrak{N}$, $\mathfrak{A}^*$).

FAP and BSS Computability on R

- Let $\mathfrak{R}$ = (R, 0, 1, +, -, ×, ÷, =, <).
- FAP($\mathfrak{R}$) = the BSS finite case partial computable functions, and FAPS($\mathfrak{R}$) = FAPCS($\mathfrak{R}$) = the BSS infinite case partial computable functions (Friedman and Mansfield 1992).
- Generalizations to arbitrary rings and fields, ordered or not, but always with the = relation.

"While" Computability on First Order Structures

- "While" schemata for computability on arbitrary first order structures (Tucker and Zucker 2000). Relations are treated as boolean valued functions.
- "While" schemata $S, S', \ldots$; 'b' for Boolean terms, 't' for individual terms built from variables and a structure's constants and functions:
- $S ::= \mathrm{skip} \mid x := t \mid S; S' \mid \mathrm{if}\ b\ \mathrm{then}\ S\ \mathrm{else}\ S' \mid \mathrm{while}\ b\ \mathrm{do}\ S.$

While Partial Computable Functions

- While($\mathfrak{A}$) = the partial functions on the domains of $\mathfrak{A}$ computable by While schemata
- WhileC($\mathfrak{A}$) = While($\mathfrak{A}$,$\mathfrak{N}$), While*($\mathfrak{A}$)=While($\mathfrak{A}$,$\mathfrak{N}$,$\mathfrak{A}$*)
- Then While($\mathfrak{A}$) = FAP($\mathfrak{A}$), WhileC($\mathfrak{A}$) = FAPC($\mathfrak{A}$), and While*($\mathfrak{A}$) = FAPCS($\mathfrak{A}$)
- Generalized Church-Turing Thesis.

"While" on Topological Partial Structures

- On structures A with a topology, the boolean valued functions of = and (e.g. on R) < are discontinuous, so must be replaced by partial functions, undefined at (x, y) when x = y.
- Defn. of effectively uniform While and While* computable functions on metric A.
- Equivalence with S-effective approximation computability.

LFP Recursion on Arbitrary Structures

- The While and While* approach covers BSS computability on R, and S-eff. approx. computability on R via metric structures.
- The general theory of LFP recursion does the same by going to type 2 schemata over arbitrary structures, without invoking topology.
- Goes back to Platek (1966), Moschovakis (1984, 1989)

Abstract Computation Procedures

- Abstract Computation Procedures (ACPs), (Feferman 1992); should have been called Abstract Recursion Procedures.
- Here structures are specified by (possibly) many-sorted domains, individual constants, partial functions, and partial monotonic functionals of type level 2.

ACP Computable Functions and Functionals

- The ACP schemata are given by Explicit Definition in type levels 1 and 2, and LFP Recursion in type 2.
- ACP($\mathfrak{A}$) = the set of partial functions over $\mathfrak{A}$ generated by the ACP schemata.
- ACP*($\mathfrak{A}$) = ACP($\mathfrak{A}$, $\mathfrak{N}$, $\mathfrak{A}^*$)

Relations to the Other Approaches

- While($\mathfrak{A}$) = ACP($\mathfrak{A}$) and While*($\mathfrak{A}$) = ACP*($\mathfrak{A}$) by Xu and Zucker 2005.
- So BSS finite and infinite dim. computable fns. on R are subsumed under the ACP approach.
- The type 2 functionals generated in ACP($\mathfrak{N}$) are just the partial recursive functionals, so the S-eff.approx. approach is also subsumed under the ACP approach.

Extensional/Intensional Aspects

- The foregoing theories are all extensional.
- ACP($\mathfrak{N}$) can also be given an intensional interpretation by replacing the partial functions and functionals by Gödel numbers.
- Each type 2 functional in this interpretation of ACP($\mathfrak{N}$) is an effective operator in the Myhill-Shepherdson sense.
- Actual computers can actually compute on codes.

Selected References

- L. Blum (2004), Computability over the reals: Where Turing meets Newton, *Notices AMS* 51, 1024-1034.
- L. Blum, F. Cuker, M. Shub and S. Smale (1997), *Complexity and Real Computation.*
- M. Braverman and S. Cook (2006), Computing over the reals: Foundations of scientific computing, *Notices AMS* 53, 318-329.

Selected References (cont'd)

- S. Feferman (1992), A new approach to computation over abstract data types, II, Lecture Notes in Comp. Sci. 626, 79-95.
- S. Feferman (t.a.), About and around computing over the reals, to appear in *Computability: Gödel, Church, Turing and Beyond* (J. Copeland, et al, eds.), available at http://math.stanford.edu/~feferman/papers/CompOverReals.pdf .

Selected References (cont'd)

- H. Friedman (1971), Algorithmic procedures, generalized Turing algorithms, and elementary recursion theory, in *Logic Colloquium '69* (R. O. Gandy and C.E.M. Yates, eds.)
- A. Grzegorczyck (1955), Computable functionals, *Fundamenta Mathematicae* 42, 168-202.
- K. Ko (1991), *Complexity Theory of Real Functions*.

Selected References (final)

- Y. Moschovakis (1989), The formal language of recursion, *J. Symbolic Logic* 54, 1216-1252.
- J.V. Tucker and J. I. Zucker (2000), Computable functions and semicomputable sets on many-sorted algebras, in *Handbook of Logic in Computer Science*, Vol. 5 (S. Abramsky et al., eds.).