Polynomial-Time Set Recursion

Joint work with Arnold Beckmann and Sam Buss

A central notion in Finite computation:

Polytime functions on finite strings

How can we generalise this notion to arbitrary sets? In other words:

When is a function $F : V \to V$ computable in "polynomial time"?

Consider some standard models for polynomial-time computation:

Polynomial-Time Set Recursion

1. Turing machines

Difficult to write an arbitrary set on a tape.

2. Fixed point logic

Even for finite structures, this works well only if there is an ordering. Moreover, on infinite ordered structures, it is too powerful (it goes beyond the hyperarithmetic).

3. Schemes

This works, using the work of Bellantoni-Cook!

Bellantoni-Cook Recursion

Idea behind Bellantoni-Cook recursion:

Define functions

$$f(\vec{x}/\vec{y})$$

where $\vec{x}, \vec{y}$ are finite sequences of finite (binary) strings and the values of f are finite strings

The components of $\vec{x}$ are the *Normal Inputs* and those of $\vec{y}$ the *Safe Inputs*

When performing primitive recursions, the "previous value" is placed on the Safe side:

$f(0, \vec{x}/\vec{y}) = g(\vec{x}/\vec{y})$

$f(z * i, \vec{x}/\vec{y}) = h_i(z, \vec{x}/\vec{y}, f(z, \vec{x}/\vec{y}))$, $i = 0$ or 1

Bellantoni-Cook Recursion

When composing, one is careful not to allow safe inputs to be copied onto the normal side:

$f(\vec{x}/\vec{y}) = h(k(\vec{x}/-)/l(\vec{x}/\vec{y}))$

Net effect: the depth of recursions performed are bounded not by the values obtained but by the sizes of the inputs

On normal inputs one gets exactly the polynomial-time computable functions

On safe inputs string-length is increased by only a constant amount

Bellantoni-Cook Recursion

A few examples (illustrated with numbers, not strings):

The successor function $S(x/y, z) = z + 1$ is in the class

Addition $A(x/y) = x + y$ is in the class, by a primitive recursion:

$A(0/y) = y$
$A(x + 1/y) = S(x/y, A(x/y)) = A(x/y) + 1$

$A^*(x, y/z) = A(y/z) = y + z$ is also in the class

Then multiplication $M(x, y/-) = x \times y$ is in the class, by a second primitive recursion on x:

$M(0, y/-) = 0$
$M(x + 1, y/-) = A^*(x, y/M(x, y/-)) = y + M(x, y/-)$

Bellantoni-Cook Recursion

BUT exponentiation is *not* in the class!

To run the previous argument for exponentiation one would need

$M^*(x, y/z) = M(y/z) = y \times z$

in the class; but we only have

$M(y, z/-)$ (multiplication of Normal Inputs)

and no function

$M(y/z) = y \times z,$

which has z as a Safe Input.

The Safe-Recursive Set Functions

We adapt the Bellantoni-Cook idea to set theory by taking the Gandy-Jensen *rudimentary set functions* as basic, and then closing under the set-theoretic analogue of Bellantoni-Cook safe recursion

Basic Functions (rudimentary set functions)

$\Pi_i^{m,n}(x_1, \ldots, x_m / x_{m+1}, \ldots, x_{m+n}) = x_i \ (1 \leq i \leq m+n)$
$\mathrm{Pair}(-/a, b) = \{a, b\}$
$\mathrm{Diff}(-/a, b) = a \setminus b$
$f(\vec{x}/\vec{a}, y) = \cup\{g(\vec{x}/\vec{a}, z) \mid z \in y\}$

Safe Recursion

$f(y, \vec{x}/\vec{a}) = h(y, \vec{x}/\vec{a}, \{f(z, \vec{x}/\vec{y}) \mid z \in y\}$

Safe Composition

$f(\vec{x}/\vec{a}) = h(\vec{r}(\vec{x}/-)/\vec{s}(\vec{x}, \vec{a}))$

The Safe-Recursive Set Functions

Some examples of SR set functions:

$S(-/a) = a \cup \{a\}$ is SR
$[f_0(-/a) = \cup\{\Pi_1^{0,1}(-/b) \mid b \in a\}$
$f_1(-/a, b) = \mathrm{Pair}(-/a, \mathrm{Pair}(-/b, b))$
$S(-/a) = f_0(-/f_1(-/a, a))]$

$S^*(a/b, c) = b \cup c$ is SR
$[S^*(a/b, c) = \cup\{\Pi_1^{0,1}(-/d) \mid d \in$
$\mathrm{Pair}(-/\Pi_2^{1,2}(a/b, c), \Pi_3^{1,2}(a/b, c))\}]$

$\oplus(a/b) = \{\oplus(c/b) \mid c \in a\} \cup b$ is SR:
$\oplus(a/b) = S^*(a/b, \{\oplus(c/b) \mid c \in a\})$
For ordinals α, β, $\oplus(\alpha/\beta) = \beta + \alpha$

The Safe-Recursive Set Functions

In analogy to getting multiplication from addition through safe-recursion, we have:
$\otimes(a, b/-) = \oplus(b/\{\otimes(c, b) \mid c \in a\}$ is SR

For ordinals α, β, $\otimes(\alpha, \beta/-) = \beta \times \alpha$

But ordinal exponentiation is *not* SR:

Proposition

If $f(\vec{x}/\vec{y})$ is a safe-recursive set function then there is a polynomial function p_f on ordinals such that:

$$rank(f(\vec{x}/\vec{y})) \leq \max(rank(\vec{y})) + p_f(rank(\vec{x}))$$

The Safe-Recursive Set Functions

How powerful are the SR (safe-recursive) functions?

Following Jensen, define:

SR-closure(A) $=$ least SR-closed $B \supseteq A$
For transitive T, SR(T) $=$ SR-closure($T \cup \{T\}$)

We have:

The Safe-Recursive Set Functions

Theorem

For transitive T, $SR(T) = L^T_{rank(T)^\omega}$, where L^T is the L-hierarchy relativised to T.

$SR(T) \supseteq L^T_{rank(T)^\omega}$:

$\vec{x} \mapsto \max(\text{rank}(\vec{x}))$ is SR

Using $\oplus, \otimes$:

$\vec{x} \mapsto \max(\text{rank}(\vec{x}))^n$ is SR for any finite n

$\vec{x} \mapsto f(\vec{x})$ is SR if $f(\vec{x})$ results from a rudimentary recursion of length $\max(\text{rank}(\vec{x}))^n$ for some finite n

Jensen: There is a rudimentary S such that for each α, J^T_α results by starting with T and iterating S $(\omega \times \alpha)$-many times

So SR(T) contains J^T_α for $\alpha < \text{rank}(T)^\omega$ and therefore contains $J^T_{rank(T)^\omega} = L^T_{rank(T)^\omega}$

The Safe-Recursive Set Functions

Conversely, $\mathrm{SR}(T) \subseteq L^T_{rank(T)^\omega}$:

Recall: For safe recursive $f(\vec{x}/\vec{y})$,
$\mathrm{rank}(f(\vec{x}/\vec{y})) \leq \max(\mathrm{rank}(\vec{y})) + p_f(\mathrm{rank}(\vec{x}))$

This also holds with "rank" replaced by "L^T-rank".
So $L^T_{rank(T)^\omega}$ is closed under SR functions.

So we conclude: $\mathrm{SR}(T) = L^T_{rank(T)^\omega}$ for transitive T

Characterisation of Safe-Recursive Set Functions

Similarly we have a characterisation of SR functions in terms of definability. For any $\vec{x}$ let $\mathrm{TC}(\vec{x})$ be the transitive closure of $\vec{x}$. The function $\vec{x} \mapsto \mathrm{TC}(\vec{x})$ is SR. Also define:

$\mathrm{SR}(\vec{x}) = \mathrm{SR}(\mathrm{TC}(\vec{x})) = L^{TC(\vec{x})}_{rank(\vec{x})^\omega}$

$\mathrm{SR}_n(\vec{x}) = L^{TC(\vec{x})}_{rank(\vec{x})^n}$ for finite n

Theorem

Suppose that $f(\vec{x}/-)$ is SR. Then for some Σ_1 formula φ and some finite n we have:

$$f(\vec{x}, -) = y \text{ iff } SR_n(\vec{x}) \models \varphi(\vec{x}, y).$$

Conversely, any function so defined is SR.

The SR Hierarchy

Analog of Jensen's J-hierarchy:

$\mathrm{SR}_1 = \mathrm{HF}$, the collection of hereditarily finite sets
$\mathrm{SR}_{\alpha+1} = \mathrm{SR}(\mathrm{SR}_\alpha)$ for $\alpha > 0$
$\mathrm{SR}_\lambda = \bigcup_{\alpha<\lambda} \mathrm{SR}_\alpha$ for limit λ

Corollary

For every α, $SR_{1+\alpha} = L_{\omega^{1+\omega\times\alpha}}$.

$L_\omega \subseteq L_{\omega^\omega} \subseteq L_{\omega^{(\omega^2)}} \subseteq L_{\omega^{(\omega^3)}} \subseteq \cdots$

Safe-Recursion on Restricted Inputs

Binary strings of length ω

If $\vec{x}$ is a finite sequence of binary ω-strings then of course $\mathrm{rank}(\vec{x})$ is less than $\omega + \omega$ so we simply have $\mathrm{SR}(\vec{x}) = L_{\omega^\omega}[\vec{x}]$.
Thus the SR functions on ω-strings look like:

$$f(\vec{x}, -) = y \textit{ iff } L_{\omega^n}[\vec{x}] \models \varphi(\vec{x})$$

where φ is Σ_1 and n is finite.

These functions coincide with those computable by an infinite-time Turing machine in time ω^n for some finite n, and were considered by Deolalikar, Hamkins, Schindler, Welch and others.

Safe-Recursion on Restricted Inputs

Finite strings
There are many ways to code finite strings as sets

A natural coding:
$\#(i * s) =$ the ordered pair $(i, \#(s)) = \{\{i\}, \{i, \#(s)\}\}$

Theorem

(Beckmann-Buss) With the above coding, the SR functions on finite strings are exactly those in the Berman class $STA(*, Exp, Poly)$ of functions which are computable by an alternating Turing machine running in exponential time with polynomially many alternations.

Interestingly, a similar class arises as the complexity of the first-order theory of the reals with $+$ and $<$ $(\mathrm{STA}(*, \mathrm{Exp}, \mathrm{Linear}))$

A Parallel-Machine Model

A machine model for Safe Recursion is possible, using processors running in parallel:

To each set x associate a processor M_x, which computes in ordinal stages. $M_x^\alpha =$ the value computed by M_x at stage α

The computation is determined by a rudimentary function h as follows:

$M_x^\alpha = h(\{(y, \beta, M_y^\beta) \mid y \in x,\ \beta \leq \alpha\} \cup \{(x, \beta, M_x^\beta) \mid \beta < \alpha\})$

The machine $\mathcal{M}$ specifies both h and a finite n; the function computed by $\mathcal{M}$ is given by:

$$f(x) = M_x^{rank(x)^n} \text{ for each } x$$

Fact: The safe recursive set functions are exactly those computable by some machine as above