Safe Recursive Set Functions

Arnold Beckmann[*] [†]
Department of Computer Science
College of Science
Swansea University
Swansea SA2 8PP, UK
a.beckmann@swansea.ac.uk

Samuel R. Buss[‡]
Department of Mathematics
University of California, San Diego
La Jolla, CA 92093-0112, USA
sbuss@math.ucsd.edu

Sy-David Friedman[§]
Kurt G??del Research Center for Mathematical Logic
University of Vienna
A-1090 Vienna, Austria
sdf@logic.univie.ac.at

March 11, 2012

Abstract

We introduce the *safe recursive set functions* based on a Bellantoni-Cook style subclass of the primitive recursive set functions. We show that under a natural encoding of finite strings by hereditarily finite sets, the functions computed by safe recursive set functions are exactly the functions computed by alternating exponential time Turing machines with polynomially many alternations.

We characterise the safe recursive set functions on arbitrary sets in definability-theoretic terms. In its strongest form, we show that a function on arbitrary sets is safe recursive if, and only if, it is uniformly definable in some polynomial level of a refinement of Jensen's J-hierarchy, relativised to the transitive closure of the function's arguments.

We observe that safe-recursive functions on infinite binary strings are equivalent to functions computed by so-called infinite-time Turing machines in time less than ω^ω. We also give a machine model for safe recursion which is based on set-indexed parallel processors and the natural bound on running times.

1 Safe Recursive Set Functions

We consider a subclass of the primitive recursive set functions [**?**]. Inspired by Bellantoni and Cook's characterization of the polynomial time computable functions [**?**], we divide arguments

[*]All three authors thank the John Templeton Foundation, Project #13152, for supporting their participation in the CRM Infinity Project at the Centre de Recerca Matemàtica, Barcelona, Catalonia, Spain during which this project was instigated.

[†]This research was partially done while the author was a visiting fellow at the Isaac Newton Institute for the Mathematical Sciences in the programme "Semantics & Syntax".

[‡]Supported in part by NSF grants DMS-0700533 and DMS-1101228, and by a grant from the Simons Foundation (#208717 to Sam Buss).

[§]Supported in part by the FWF (Austrian Science Fund) through FWF project number P 22430-N13.

of set functions into normal and safe ones. By writing $f(\vec{x} \,/\, \vec{a})$ we indicate that $\vec{x}$ are f's normal arguments, and $\vec{a}$ its safe arguments. Bellantoni and Cook use the notation $f(\vec{x};\vec{a})$ instead of $f(\vec{x} \,/\, \vec{a})$, using semicolon (;) instead of slash (/). We use the slash instead, as we find it improves readability. Set functions whose arguments are typed in this way will be denoted *safe set functions.*

1.1 Safe Rudimentary Set Functions

We first define *safe rudimentary set functions* based on rudimentary set functions [**?**].

Definition 1.1. The set of *safe rudimentary set functions (sRud)* is the smallest class of safe set functions that contains the initial functions (i)–(iii) and is closed under *bounded union* (iv) and *safe composition* (v):

(i) **(Projection)** $\pi_j^{n,m}(x_1,\dots,x_n \,/\, x_{n+1},\dots,x_{n+m}) = x_j$, for $1 \le j \le n+m$, is in *sRud.*

(ii) **(Difference)** $\mathrm{d}(/\, a,b) = a \setminus b$ is in *sRud.*

(iii) **(Pairing)** $\mathrm{p}(/\, a,b) = \{a,b\}$ is in *sRud.*

(iv) **(Bounded Union)** If g is in *sRud*, then

$$f(\vec{x} \,/\, \vec{a}, b) = \bigcup_{z\in b} g(\vec{x} \,/\, \vec{a}, z)$$

is in *sRud.*

(v) **(Safe Composition)** If $h, \vec{r}, \vec{t}$ are in *sRud*, then

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

is in *sRud.*

We list a few functions which are definable in *sRud.* Details of the definitions of some of these can also be found in [**?**]. Let (a,b) denote Kuratowski's ordered pair $\{\{a\},\{a,b\}\}$. The functions pr_ℓ and pr_r extract the first and second element from an ordered pair.

- $\mathrm{Union}(/\, a) = \cup a$ and $\mathrm{Intersec}(/\, a,b) = a \cap b$.
  $\mathrm{Union}(/\, a) = \bigcup_{z\in a} \pi_1^{0,1}(/\, z)$ and $\mathrm{Intersec}(/\, a,b) = c \setminus ((c\setminus a) \cup (c \setminus b))$ for $c = a \cup b$.

- $\mathrm{Succ}(/\, a) = a \cup \{a\}$, $\mathrm{kop}(/\, a,b) = (a,b)$, $\mathrm{pr}_\ell(/\, (a,b)) = a$, $\mathrm{pr}_r(/\, (a,b)) = b$.
  $f(/\, c) = \bigcup_{z\in c}\bigcup_{y\in c}(z\setminus y)$ satisfies $f(/\, (a,b)) = \begin{cases}\{b\} & \text{if } a \neq b\\ \emptyset & \text{otherwise}\end{cases}$,
  thus $\mathrm{pr}_\ell(/\, c) = \cup(\cup c \setminus f(/\, c))$.
  $g(/\, c) = \cup(c \setminus \{\cup c\})$ satisfies $g(/\, (a,b)) = \begin{cases}\{a\} & \text{if } a \neq b\\ \emptyset & \text{otherwise}\end{cases}$, thus $\mathrm{pr}_r(/\, c) = \cup(\cup c \setminus g(/\, c))$.

- $\mathrm{Cond}_=(/\, a,b,c,d) = \begin{cases} a & \text{if } c = d\\ b & \text{otherwise.}\end{cases}$
  Let $\bar{g}(\,/\, a,c,z) = \bigcup\{a\colon u \in c\setminus z \cup z\setminus c\}$ and $g(\,/\, a,c,z) = a \setminus \bar{g}(\,/\, a,c,z)$,
  then $\bar{g}(\,/\, a,c,z) = \begin{cases} a & \text{if } z \neq c\\ \emptyset & \text{otherwise}\end{cases}$ and $g(\,/\, a,c,z) = \begin{cases} a & \text{if } z = c\\ \emptyset & \text{otherwise.}\end{cases}$
  Thus $\mathrm{Cond}_=(\,/\, a,b,c,d) = g(/\, a,c,d) \cup \bar{g}(/\, b,c,d)$.

- $\mathrm{Cond}_{\in}(/\, a, b, c, d) = \begin{cases} a & \text{if } c \in d \\ b & \text{otherwise.} \end{cases}$

  Let $h(\,/\, a, c, d) = \bigcup\{g(/\, a, c, z)\colon z \in d\}$ (g as defined for $\mathrm{Cond}_=$) and $\bar{h}(\,/\, b, c, d) = b \setminus h(/\, b, c, d)$, then $h(\,/\, a, c, d) = \begin{cases} a & \text{if } c \in d \\ \emptyset & \text{otherwise} \end{cases}$ and $\bar{h}(\,/\, b, c, d) = \begin{cases} b & \text{if } c \notin d \\ \emptyset & \text{otherwise.} \end{cases}$
  Thus $\mathrm{Cond}_{\in}(/\, a, b, c, d) = h(/\, a, c, d) \cup \bar{h}(/\, b, c, d)$.

- $\mathrm{Appl}(/\, a, b) = \{y\colon (\exists x \in b)(x, y) \in a\}$.
  Let $g(/\, b, c) = \begin{cases} \{\mathrm{pr}_r(c)\} & \text{if } \mathrm{pr}_\ell(c) \in b \\ \emptyset & \text{otherwise} \end{cases}$, then $\mathrm{Appl}(/\, a, b) = \bigcup\{g(/\, b, c)\colon c \in a\}$.

- $\mathrm{Prod}(/\, a, b) = \{(x, y)\colon x \in a, y \in b\} =: a \times b$, by first observing that

$$f(/\, x, b) = \{(x, y)\colon y \in b\} = \bigcup \big\{\{(x, y)\}\colon y \in b\big\}$$

  is in *sRud*, and then that $\mathrm{Prod}(/\, a, b) = \bigcup \big\{f(x, b)\colon x \in a\big\}$.

1.2 Predicative Set Recursion

We extend the safe rudimentary set function by a predicative set recursion scheme.

Definition 1.2. The set of *safe recursive set functions (SRSF)* is the smallest class which contains the safe rudimentary set functions and is closed under safe composition, bounded union and the following scheme:
(Predicative Set Recursion) If h is in *SRSF*, then

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

is in *SRSF*. Observe that according to our convention for denoting functions, x is a normal argument of f, and $\{f(z, \vec{y}\,/\,\vec{a})\colon z \in x\}$ is substituted at a safe argument of h.

We show that ordinal addition and multiplication are in *SRSF*. We will see later that ordinal exponentiation cannot be defined in *SRSF*. In a set context, let $0, 1, 2, \ldots$ denote ordinals in the usual sense, e.g., $0 = \emptyset$ and $1 = \{\emptyset\}$.

- $\mathrm{Add}(x\,/\,a) = \begin{cases} a & \text{if } x = 0 \\ \mathit{Succ}(/\, \bigcup\{\mathrm{Add}(z\,/\,a)\colon z \in x\}) & \text{if } x = \mathit{Succ}(/\, \bigcup x) \\ \bigcup\{\mathrm{Add}(z\,/\,a)\colon z \in x\} & \text{otherwise.} \end{cases}$ $\alpha{+}\beta := \mathrm{Add}(\beta\,/\,\alpha)$
  satisfies the usual recursive equations for ordinal addition. Observe that for $\alpha + \beta$, β is a normal argument and α a safe argument.

- $\mathrm{Mult}(x, y\,/) = \begin{cases} 0 & \text{if } x = 0 \\ \mathrm{Add}(y\,/\, \bigcup\{\mathrm{Mult}(z, y\,/)\colon z \in x\}) & \text{if } x = \mathit{Succ}(/\, \bigcup x) \\ \bigcup\{\mathrm{Mult}(z, y\,/)\colon z \in x\} & \text{otherwise.} \end{cases}$ $\alpha{\cdot}\beta := \mathrm{Mult}(\beta, \alpha\,/)$
  satisfies the usual recursive equations for ordinal multiplication. Observe that for $\alpha \cdot \beta$, both α and β are normal.

It should be pointed out here that as Mult has no safe arguments we cannot similarly define exponentiation via predicative set recursion, as we did for Add and Mult.

In many situations it will be convenient to define predicates instead of functions. In the following we provide the necessary background for this.

Definition 1.3 (Predicates)**.** A predicate $R(\vec{x} \,/\, \vec{a})$ is in *SRSF* (in *sRud*, resp.) if the function

$$\chi_R(\vec{x} \,/\, \vec{a}) = \begin{cases} 1 & \text{if } R(\vec{x} \,/\, \vec{a}) \\ 0 & \text{otherwise} \end{cases}$$

is in *SRSF* (in *sRud*, resp.) Remember that 0 and 1 in a set theoretic context denote ordinals.

Examples of predicates in *sRud* are $a \in b$, $a \notin b$, $a = b$, and $a \neq b$ for safe a, b, which can be seen using the safe rudimentary functions $\mathrm{Cond}_\in$ and $\mathrm{Cond}_=$ as provided before.

Predicates can be used to define functions by separation in the usual way. E.g., assume $R(\vec{x} \,/\, \vec{a}, b)$ is a predicate in *SRSF*, and $B(\vec{x} \,/\, \vec{a})$ a function in *SRSF*. Then $f(\vec{x} \,/\, \vec{a}) = \{b \in B(\vec{x} \,/\, \vec{a}) \colon R(\vec{x} \,/\, \vec{a}, b)\}$ is a function in *SRSF*. To see this, let

$$sel(\vec{x} \,/\, \vec{a}, b) \quad = \quad \begin{cases} \{b\} & \text{if } R(\vec{x} \,/\, \vec{a}, b) \\ \emptyset & \text{otherwise} \end{cases} \quad = \quad \mathrm{Cond}_=(/\, \emptyset, \{b\}, \chi_R(\vec{x} \,/\, \vec{a}, b), 0) \ .$$

Then $f(\vec{x} \,/\, \vec{a})$ can be defined by bounded union as $\bigcup_{b \in B(\vec{x} \,/\, \vec{a})} sel(\vec{x} \,/\, \vec{a}, b)$.

Proposition 1.4 (Closure Properties of Predicates)**.** *Predicates in SRSF (in sRud, resp.) are closed under Boolean operations and bounded quantification over safe arguments.*

Proof. Let Q, Q_1 and Q_2 be predicates in *SRSF* (in *sRud*, resp.). Then $\neg Q_1(\vec{x} \,/\, \vec{a})$, $Q_1(\vec{x} \,/\, \vec{a}) \vee Q_2(\vec{x} \,/\, \vec{a})$ and $(\exists c \in a_1) Q(\vec{x} \,/\, \vec{a}, c)$ are predicates in *SRSF* (in *sRud*, resp.):

- $P(\vec{x} \,/\, \vec{a}) \Leftrightarrow \neg Q_1(\vec{x} \,/\, \vec{a})$ can be defined as $\chi_P(\vec{x} \,/\, \vec{a}) = \{\emptyset\} \setminus \chi_{Q_1}(\vec{x} \,/\, \vec{a})$.
- $P(\vec{x} \,/\, \vec{a}) \Leftrightarrow Q_1(\vec{x} \,/\, \vec{a}) \vee Q_2(\vec{x} \,/\, \vec{a})$ can be defined as

$$\chi_P(\vec{x} \,/\, \vec{a}) = \mathrm{Cond}_\in \Big(/\, 1, 0, 1, \big\{\chi_{Q_1}(\vec{x} \,/\, \vec{a}), \chi_{Q_2}(\vec{x} \,/\, \vec{a})\big\}\Big) \ .$$

- $P(\vec{x} \,/\, \vec{a}) \Leftrightarrow (\exists c \in a_1) Q(\vec{x} \,/\, \vec{a}, c)$ can be defined as

$$\chi_P(\vec{x} \,/\, \vec{a}) = \mathrm{Cond}_\in \Big(/\, 1, 0, 0, \bigcup_{c \in a_1} \chi_Q(\vec{x} \,/\, \vec{a}, c)\Big) \ .$$

□

Further examples of predicates in *sRud* are $\mathrm{trans}(/\, a)$ (a is transitive) and $\mathrm{Ord}(/\, a)$ (a is an ordinal.) This can be seen using the previous proposition:

$$\begin{aligned} \mathrm{trans}(/\, a) \quad &\Leftrightarrow \quad \forall b \in a \ \forall c \in b \ c \in a \\ \mathrm{Ord}(/\, a) \quad &\Leftrightarrow \quad \mathrm{trans}(/\, a) \wedge \forall b \in a \ \mathrm{trans}(/\, b) \end{aligned}$$

1.3 Bounding Ranks

A very important property of safe recursive set functions is that they increase ranks only polynomially. This can be proven similarly to the corresponding Lemma 4.1 in [?]. Let $\mathrm{rk}(x) = \bigcup\{\mathrm{rk}(y) + 1 \colon y \in x\}$ denote the rank of x. Observe that $\mathrm{rk}(x\,/)$ is in *SRSF*. It should be stressed that the next theorem is *not* restricted to sets of finite rank.

Theorem 1.5. *Let f be a function in SRSF. There is a polynomial q_f such that*

$$\mathrm{rk}(f(\vec{x} \,/\, \vec{a})) \quad \leq \quad \max_i \mathrm{rk}(a_i) + q_f(\mathrm{rk}(\vec{x}))$$

for all *sets $\vec{x}$, $\vec{a}$.*

Proof. The proof is by induction on the definition of f in *SRSF*. Our construction will ensure that q_f will always be a multi-variable polynomial with coefficients given by natural numbers. This implies that it will be a *monotone* polynomial on ordinals, i.e., if any of its arguments will be increased, leaving the other arguments the same, its value does not decrease.

We will only consider the case that f is defined by predicative set recursion, the other cases (base cases, bounded union, safe composition) are left to the reader.

If $f(x, \vec{y} \,/\, \vec{a})$ is defined by predicative set recursion from h, then by induction hypothesis we have q_h bounding h. Define q_f such that

$$q_f(\alpha, \vec{\beta}) \quad = \quad (1 + q_h(\alpha, \vec{\beta})) \cdot (1 + \alpha) \enspace .$$

We will show that $\mathrm{rk}(f(x, \vec{y} \,/\, \vec{a})) \leq \max\{\mathrm{rk}(\vec{a})\} + q_f(\mathrm{rk}(x), \mathrm{rk}(\vec{y}))$ by $\in$-induction on x.

$$\begin{aligned}
&\mathrm{rk}(f(x, \vec{y} \,/\, \vec{a})) \\
&\quad = \mathrm{rk}\left(h(x, \vec{y} \,/\, \vec{a}, \{f(z, \vec{y} \,/\, \vec{a})\colon z \in x\})\right) \\
&\quad \leq \max\left\{ \mathrm{rk}(\vec{a}), \mathrm{rk}\left(\{f(z, \vec{y} \,/\, \vec{a})\colon z \in x\} \right) \right\} + q_h(\mathrm{rk}(x), \mathrm{rk}(\vec{y})) \\
&\quad = \max\left\{ \mathrm{rk}(\vec{a}), \bigcup \left\{ \mathrm{rk}(f(z, \vec{y} \,/\, \vec{a})) + 1\colon z \in x \right\} \right\} + q_h(\mathrm{rk}(x), \mathrm{rk}(\vec{y})) \\
&\quad \leq \max\left\{ \mathrm{rk}(\vec{a}), \bigcup \left\{ \max\{\mathrm{rk}(\vec{a})\} + q_f(\mathrm{rk}(z), \mathrm{rk}(\vec{y})) + 1\colon z \in x \right\} \right\} + q_h(\mathrm{rk}(x), \mathrm{rk}(\vec{y})) \\
&\quad = \max\{\mathrm{rk}(\vec{a})\} + \bigcup \left\{ q_f(\mathrm{rk}(z), \mathrm{rk}(\vec{y})) + 1\colon z \in x \right\} + q_h(\mathrm{rk}(x), \mathrm{rk}(\vec{y})) \\
&\quad = \max\{\mathrm{rk}(\vec{a})\} + \bigcup \left\{ q_f(\mathrm{rk}(z), \mathrm{rk}(\vec{y})) + 1 + q_h(\mathrm{rk}(x), \mathrm{rk}(\vec{y}))\colon z \in x \right\}
\end{aligned}$$

where for the second "$\leq$" we used the $\in$-induction hypothesis. Let α be $\mathrm{rk}(x)$, β_i be $\mathrm{rk}(y_i)$, and γ be $\mathrm{rk}(z)$. Assume $\gamma < \alpha$, then we will show that

$$q_f(\gamma, \vec{\beta}) + 1 + q_h(\alpha, \vec{\beta}) \leq q_f(\alpha, \vec{\beta}) \enspace . \tag{1.1}$$

Using this we can continue our calculation showing

$$\mathrm{rk}(f(x, \vec{y} \,/\, \vec{a})) \leq \max\{\mathrm{rk}(\vec{a})\} + q_f(\mathrm{rk}(x), \mathrm{rk}(\vec{y})) \enspace .$$

We finish by proving (1.1):

$$\begin{aligned}
q_f(\gamma, \vec{\beta}) + 1 + q_h(\alpha, \vec{\beta}) &= (1 + q_h(\gamma, \vec{\beta})) \cdot (1 + \gamma) + 1 + q_h(\alpha, \vec{\beta}) \\
&\leq (1 + q_h(\alpha, \vec{\beta})) \cdot (1 + \gamma) + 1 + q_h(\alpha, \vec{\beta}) \\
&= (1 + q_h(\alpha, \vec{\beta})) \cdot (1 + \gamma + 1) \\
&\leq (1 + q_h(\alpha, \vec{\beta})) \cdot (1 + \alpha) \\
&= q_f(\alpha, \vec{\beta}) \enspace .
\end{aligned}$$

□

Corollary 1.6. *Ordinal exponentiation cannot be computed by a safe recursive set function.*

2 Computing on Hereditarily Finite Sets

For this section, we *restrict our attention to the set $\mathbb{H}F$ of hereditarily finite sets only.* Our main result for $\mathbb{H}F$ is that the *SRSF* functions acting on $\mathbb{H}F$ can be characterized in terms of ATIME$(2^{n^{O(1)}}, n^{O(1)})$; namely, the class of predicates computable by an alternating Turing machine which runs in time $2^{n^{O(1)}}$ with up to $n^{O(1)}$ many alternations. It is interesting to note that this complexity class is known to characterize the decision problem for the theory of the reals with addition. In particular, the theory of the reals with addition is many-one complete for ATIME$(2^{n^{O(1)}}, n^{O(1)})$ under polynomial time reductions [**?**, **?**, **?**].

On $\mathbb{H}F$ we will often drive a recursion by some special sets which we denote *skinny drivers.* We define the *skinny drivers of rank n*, sd_n, by induction on n as follows: $\mathrm{sd}_0 = \emptyset$ and $\mathrm{sd}_{n+1} = \{\mathrm{sd}_n\}$. Turning our attention to skinny drivers on $\mathbb{H}F$ is not a restriction, as the function $\mathrm{sd}(x\,/) = \mathrm{sd}_{\mathrm{rk}(x)}$ is in *SRSF*, which can be seen as follows:

$$\mathrm{sd}(x\,/) = \overline{\mathrm{sd}}(\mathrm{rk}(x\,/)\,/) \qquad \overline{\mathrm{sd}}(\alpha\,/) = h(/\ \{\overline{\mathrm{sd}}(\beta)\colon \beta \in \alpha\})$$

$$h(/\,b) = \bigcup_{z\in b} g(/\,z, \bigcup b) \qquad g(/\,z,c) = \begin{cases} \emptyset & \text{if } z \in c \\ \{z\} & \text{otherwise} \end{cases}$$

Predicative set recursion based on skinny drivers can be written in a simplified way.

Proposition 2.1 (Skinny Predicative Set Recursion)**.** *Let g, h be in SRSF of appropriate arities. Then there exists some f in SRSF which satisfies*

$$\begin{aligned} f(\emptyset, \vec{y}\,/\,\vec{a}) &= g(\vec{y}\,/\,\vec{a}) \\ f(\{d\}, \vec{y}\,/\,\vec{a}) &= h(\{d\}, \vec{y}\,/\,\vec{a}, f(d, \vec{y}\,/\,\vec{a}))\ . \end{aligned}$$

Proof. Let

$$H(x, \vec{y}\,/\,\vec{a}, b) = \begin{cases} g(\vec{y}\,/\,\vec{a}) & \text{if } x = \emptyset \\ h(x, \vec{y}\,/\,\vec{a}, \bigcup b) & \text{otherwise.} \end{cases}$$

Then f defined by predicative set recursion on x in H satisfies the required equations. $\square$

In the previous subsection we have seen one important property of *SRSF* that ranks of sets grow polynomially only. Another important property deals with sizes of sets, in particular their growth rate. Since there are super-exponentially many sets of rank n, Theorem 1.5 implies a super-exponential bound on the size of the transitive closure of $f(\vec{x}\,/\,\vec{a})$ for $f \in$ *SRSF*. The following Theorem 2.3 will give a substantial improvement over this by showing a double exponential upper bound. Functions which satisfy such a double exponential size upper bound will be called *dietary* – the following definition will make this notion precise.

Let $|a|$ denote the cardinality of a set a, and $\mathrm{tc}(a)$ its transitive closure.

Definition 2.2. A function $f(\vec{x}\,/\,\vec{a})$ in *SRSF* is called *dietary* if for some polynomial p,

$$|\,\mathrm{tc}(f(\vec{x}\,/\,\vec{a}))| \quad \le \quad |\,\mathrm{tc}(\{\vec{x}, \vec{a}\})|^{2^{p(\mathrm{rk}(\vec{x}))}}$$

for all $\vec{x}, \vec{a} \in \mathbb{H}F$.

Theorem 2.3. *All functions in SRSF are dietary.*

Proof. The proof is by induction on the definition of f in *SRSF*. We will construct *monotone* polynomials q_f, and show that they can serve as the polynomial p in the bound of the assertion that f is dietary. We will only consider the case that f is defined by predicative set recursion, the other cases (base cases, bounded union, safe composition) are left to the reader.

If f is defined by predicative set recursion from h, then by induction hypothesis we have h dietary with bounding polynomial q_h. Define q_f such that

$$q_f(\alpha, \vec{\beta}) \quad = \quad (1 + q_h(\alpha, \vec{\beta})) \cdot (1 + \alpha) \ .$$

We will show that $|f(x, \vec{y} \,/\, \vec{a})| \leq |\operatorname{tc}(x, \vec{y}, \vec{a})|^{2^{q_f(\operatorname{rk}(x), \operatorname{rk}(\vec{y}))}}$ by $\in$-induction on x. We have

$$\begin{aligned}
|f(x, \vec{y} \,/\, \vec{a})| &= |h(x, \vec{y} \,/\, \vec{a}, \{f(z, \vec{y} \,/\, \vec{a}) \colon z \in x\})| \\
&\leq |\operatorname{tc}\left(\{x, \vec{y}, \vec{a}, \{f(z, \vec{y} \,/\, \vec{a}) \colon z \in x\}\}\right)|^{2^{q_h(\operatorname{rk}(x), \operatorname{rk}(\vec{y}))}} \\
&\leq \Big(|\operatorname{tc}(\{x, \vec{y}, \vec{a}\})| + \sum_{z \in x} |\operatorname{tc}(f(z, \vec{y}, \vec{a}))| + |x| + 1\Big)^{2^{q_h(\operatorname{rk}(x), \operatorname{rk}(\vec{y}))}}
\end{aligned}$$

Let α be $\operatorname{rk}(x)$ and β_i be $\operatorname{rk}(y_i)$. For $z \in x$ we compute, using $\in$-induction hypothesis,

$$|\operatorname{tc}(f(z, \vec{y} \,/\, \vec{a}))| \ \leq \ |\operatorname{tc}(\{x, \vec{y}, \vec{a}\})|^{2^{q_f(\operatorname{rk}(z), \vec{\beta})}} \ \leq \ |\operatorname{tc}(\{x, \vec{y}, \vec{a}\})|^{2^{q_f(\alpha - 1, \vec{\beta})}}$$

We continue our computation from above:

$$\begin{aligned}
|f(x, \vec{y} \,/\, \vec{a})| &\leq \Big(|\operatorname{tc}(\{x, \vec{y}, \vec{a}\})| + |x| \cdot |\operatorname{tc}(\{x, \vec{y}, \vec{a}\})|^{2^{q_f(\alpha-1, \vec{\beta})}} + |x| + 1\Big)^{2^{q_h(\alpha, \vec{\beta})}} \\
&\leq \Big((|x| + 1) \cdot |\operatorname{tc}(\{x, \vec{y}, \vec{a}\})|^{2^{q_f(\alpha-1, \vec{\beta})}}\Big)^{2^{q_h(\alpha, \vec{\beta})}} \\
&\leq |\operatorname{tc}(\{x, \vec{y}, \vec{a}\})|^{2^{q_f(\alpha-1, \vec{\beta})+1} \cdot 2^{q_h(\alpha, \vec{\beta})}} \\
&\leq |\operatorname{tc}(\{x, \vec{y}, \vec{a}\})|^{2^{q_f(\alpha, \vec{\beta})}} \ .
\end{aligned}$$

For the last inequality we observe:

$$\begin{aligned}
2^{q_f(\alpha-1, \vec{\beta})+1} \cdot 2^{q_h(\alpha, \vec{\beta})} &= 2^{q_f(\alpha-1, \vec{\beta})+1+q_h(\alpha, \vec{\beta})} \\
&= 2^{(1+q_h(\alpha-1, \vec{\beta}))\cdot(1+\alpha-1)+1+q_h(\alpha, \vec{\beta})} \\
&\leq 2^{(1+q_h(\alpha, \vec{\beta}))\cdot(1+\alpha-1)+1+q_h(\alpha, \vec{\beta})} \\
&= 2^{(1+q_h(\alpha, \vec{\beta}))\cdot(1+\alpha)} = 2^{q_f(\alpha, \vec{\beta})} \ .
\end{aligned}$$

□

That the bounds given in the definition of "dietary" are sharp, can be seen in the following way. Let $\operatorname{Sq}(/\, a) = \operatorname{Prod}(/\, a, a)$. Define f by skinny predicative set recursion as follows: $f(\emptyset \,/\, a) = a$ and $f(\{d\} \,/\, a) = \operatorname{Sq}(/\, f(d \,/\, a))$. Then f is in *SRSF*, and satisfies $|f(\operatorname{sd}_n \,/\, a)| = |a|^{2^n}$.

2.1 Simulating Alternating Turing Machines

We will describe a way in which alternating Turing machine computations can be simulated in *SRSF*. An *alternating Turing machine (ATM)* is given by an 8-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject}, g)$ where the first 7 components form the ingredients of a nondeterministic Turing machine in the usual way, that is, Q is a finite set of states which includes three designated states: the start state q_0, the accepting state q_{accept}, and the rejecting state q_{reject}, Σ is the input alphabet, Γ the work tape alphabet which includes Σ and an additional symbol $\sqcup$ denoting a blank tape cell, and $\delta \subset Q \times \Gamma \times Q \times \Gamma \times \{L, R\}$ is the transition relation. In addition to this, $g\colon Q \to \{\vee, \wedge\}$ divides the set of states into universal ($\wedge$) and existential ($\vee$) states. A *configuration* is given by a 3-tuple (u, q, v) where q is a state in Q, u and v are words over Γ , which indicates the configuration where the current state is q, the tape content is uv, and the head position is the first symbol of v (the tape contains only blanks following the last symbol of v,) and the label of this configuration is given by $g(q)$.

We will not define the behavior of our ATMs in full detail, these will be obvious from the context. We do use two special conventions, however, that might lead to confusion if not stated explicitly. First, we assume that the tape is open only to the right, initially the input word is written as the first entries from the left with the head positioned at the first symbol. Second, when we mention a time bound for an ATM, then we assume that the ATM is equipped with a counter, and enters the reject state should the time bound be exceeded.

We are interested in a complexity class of alternating time with a bounded number of alternation. Given functions $t(n)$ and $q(n)$ we define the set $\mathrm{ATIME}(t(n), q(n))$ to consist of all languages which can be decided by some alternating Turing machine which runs, on inputs of length n, in time bounded by $O(t(n))$, such that the number of alternations on each computation path is bounded by $O(q(n))$.

For our simulation of an ATM $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject}, g)$ within *SRSF*, we assume that the alphabet Γ consists of sets only, and that $\emptyset \notin \Gamma$. Taking into considerations that functions in *SRSF* are dietary, and increase ranks of sets only polynomially, we will represent configurations as sets in the following way: The tape content will be encoded as a full binary tree (the *tape tree*) whose leaves are labeled with elements from Γ; and the head position will be encoded as a binary sequence (the *head path*) of length corresponding to the height of the tape tree. For this, we define the empty sequence by $\emptyset$, and in general the binary sequence $\langle i_1, \dots, i_n\rangle$ of length n by $(i_1, (i_2, \dots, (i_n, \emptyset) \dots))$. Let $\mathcal{T}_n^\Gamma$ be the set of all tape trees of height n, and $\mathcal{P}_n$ be the set of all head paths of length n. Observe that a tape tree of height n stores tapes of length 2^n. The set of all configurations of size 2^n is now given as $\mathcal{C}_n^M = Q \times \mathcal{P}_n \times \mathcal{T}_n^\Gamma$. All these sets can be defined by functions in *SRSF*: Choose $\mathcal{P}$ in *SRSF* satisfying $\mathcal{P}(\emptyset\,/) = \{\emptyset\}$ and $\mathcal{P}(\{d\}\,/) = \mathrm{Prod}(/\,\{0,1\}, \mathcal{P}(d\,/))$, then $\mathcal{P}(\mathrm{sd}_n\ /) = \mathcal{P}_n$. Choose $\mathcal{T}^M$ in *SRSF* such that $\mathcal{T}^M(\emptyset\,/) = \Gamma$ and $\mathcal{T}^M(\{d\}\,/) = \mathrm{Sq}(/\,\mathcal{T}^M(d\,/))$, then $\mathcal{T}^M(\mathrm{sd}_n\ /) = \mathcal{T}_n^\Gamma$. Define $C^M(d\,/)$ as $\mathrm{Prod}(/\,Q, \mathrm{Prod}(/\,\mathcal{P}(d\,/), \mathcal{T}^M(d\,/)))$ then $\mathcal{C}^M(\mathrm{sd}_n\ /) = \mathcal{C}_n^M$.

We define a predicate Next^M describing successor configurations according to M. $\mathrm{Next}^M(\mathrm{sd}_n\ / c, c')$ will be true if $c, c' \in \mathcal{C}_n^M$ and c' is a possible next configuration from c. It can be defined as a predicate in *SRSF* in the following way:

$$\mathrm{Next}^M(d\,/\,(q,p,t),(q',p',t')) \Leftrightarrow$$
$$\bigvee_{(q,s,q',s',o)\in\delta} \big[\mathit{Read}(d\,/\,p,t) = s \ \wedge\ \mathit{Move}_o(d\,/\,p) = p' \ \wedge\ \mathit{Prt}(d\,/\,p,t,s') = t'\big]$$

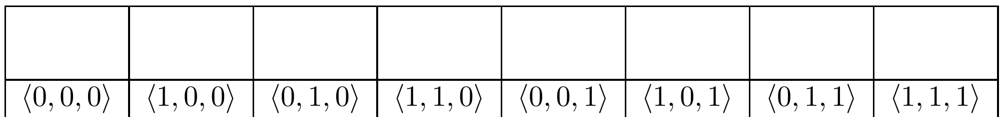

Figure 1: A tape of length 8 with pointers. Note that the cells are indexed by binary strings in reversed bit order.

where $Read(d\,/\,p,t)$ outputs the symbol on tape t at position p:

$$\begin{aligned} Read(\emptyset\,/\,p,t) &= t \\ Read(\{d\}\,/\,(i,p),(t_0,t_1)) &= Read(d\,/\,p,t_i) \ . \end{aligned}$$

$Move_o(d\,/\,p)$ computes the head position obtained by moving from position p in direction $o \in \{L, R\}$, where $\langle 0,\dots,0\rangle$ denotes the very left position (see Figure 1):

$$\begin{aligned} Move_o(\emptyset\,/\,p) &= 0 \\ Move_L(\{d\}\,/\,(i,p)) &= \begin{cases} \langle 0,\dots,0\rangle & \text{if } (i,p) = \langle 0,\dots,0\rangle \\ (0,p) & \text{if } i = 1 \\ (1, Move_L(d\,/\,p)) & \text{if } i = 0 \end{cases} \\ Move_R(\{d\}\,/\,(i,p)) &= \begin{cases} \langle 1,\dots,1\rangle & \text{if } (i,p) = \langle 1,\dots,1\rangle \\ (1,p) & \text{if } i = 0 \\ (0, Move_R(d\,/\,p)) & \text{if } i = 1 \end{cases} \end{aligned}$$

and $Prt(d\,/\,p,t,s')$ computes the tape obtained by printing the symbol s' on tape t at position p:

$$\begin{aligned} Prt(\emptyset\,/\,p,t,s) &= s \\ Prt(\{d\}\,/\,(0,p),(t_0,t_1),s) &= (Prt(d\,/\,p,t_o,s),t_1) \\ Prt(\{d\}\,/\,(1,p),(t_0,t_1),s) &= (t_0, Prt(d\,/\,p,t_1,s)) \ . \end{aligned}$$

We also need a predicate $\overline{\mathrm{Next}}^M$ describing successor configurations according to M for which the label according to g does not change. Let the labeling function g be extended to configurations in the obvious way: $g((q,p,t)) := g(q)$. $\overline{\mathrm{Next}}^M(d\,/\,c,c')$ can be defined as $\mathrm{Next}^M(d\,/\,c,c')$ and $g(c) = g(c')$.

Our next aim is to define a binary relation $\mathcal{N}_n^M$ on $\mathcal{C}_n^M$ which represents the iteration of $\overline{\mathrm{Next}}^M$. The situation of iterating a binary relation R on a set A will occur at various places, therefore we will first explain how to achieve this as a function in *SRSF*.

Given two sets r and s (think of $r \subseteq A \times B$ and $s \subseteq B \times C$) we define their composition $r \circ s$ to be the set $\{(x,z) \in A \times C\colon (\exists y \in B)(x,y) \in r \wedge (y,z) \in s\}$. This can be defined in *sRud* as $Comp(/\,r,s) = r \circ s$ because *sRud* is closed under Boolean connectives and bounded quantification. Let A and R be sets (think of R being a binary relation on A.) We define the iteration of R on A as

$$\mathrm{Iter}(d_n\,/\,R,A) = \{(x,y) \in A \times A\colon \text{there is a path in } R \text{ from } x \text{ to } y \text{ of length } \leq 2^n\}$$

which can be defined by skinny recursion in *SRSF* as follows:

$$\begin{aligned}\mathrm{Iter}(\emptyset \,/\, R, A) &= R \cup \{(x,x) \colon x \in A\}\\ \mathrm{Iter}(\{d\} \,/\, R, A) &= \mathit{Comp}(/\ \mathrm{Iter}(d \,/\, R, A), \mathrm{Iter}(d \,/\, R, A))\ .\end{aligned}$$

Let us return to our task of iterating $\overline{\mathrm{Next}}^M$. We define

$$\begin{aligned}\overline{\mathcal{N}}^M(d\,/) &= \left\{(c,c') \in \mathcal{C}^M(d\,/) \colon \overline{\mathrm{Next}}^M(d\,/\,c,c')\right\}\\ \mathcal{N}^M(d\,/) &= \mathrm{Iter}(d\,/\,\overline{\mathcal{N}}^M(d\,/), \mathcal{C}^M(d\,/))\end{aligned}$$

Thus $\overline{\mathcal{N}}_n^M = \overline{\mathcal{N}}^M(\mathrm{sd}_n\ /)$ and $\mathcal{N}_n^M = \mathcal{N}^M(\mathrm{sd}_n\ /)$.

Let $\mathrm{NEXT}^M(\mathrm{sd}_n\ /\,c,c')$ denote the predicate on configurations $c, c' \in \mathcal{C}_n^M$ which is true iff c' follows from c according to M such that either c, c' and all intermediate configurations have the same label and c' is an accepting or rejecting configuration, or c and all intermediate configurations have the same label, and c' is the first with a different label.

$$\begin{aligned}&\mathrm{NEXT}^M(d\,/\,c,(q',p',t'))\\ &\qquad \Leftrightarrow\ (\exists c'' \in \mathcal{C}^M(d\,/))\big[(c,c'') \in \mathcal{N}^M(d\,/)\ \wedge\ \mathrm{Next}^M(d\,/\,c'',(q',p',t'))\\ &\qquad\qquad \wedge\ [g(c) \neq g(q') \vee q' \in \{q_{accept}, q_{reject}\}]\big]\\ &\mathcal{NEXT}^M(d\,/)\ =\ \{(c,c') \in \mathcal{C}^M(d\,/) \colon \mathrm{NEXT}^M(d\,/\,c,c')\}\end{aligned}$$

Finally, we define the accepting states of an alternating computation according to M. Let C be a set (the set of configurations) and N a binary relation on C (taking configurations to a next alternating configuration.) $\mathrm{Accept}^M(\mathrm{sd}_n\ /\,c,C,N)$ will be true if c has an accepting computation of at most n alternations.

$$\begin{aligned}\mathrm{Accept}^M(\emptyset\,/\,c,C,N) &\quad\Leftrightarrow\quad c \in C\ \wedge\ \mathit{state}(c) = q_{accept}\\ \mathrm{Accept}^M(\{d\}\,/\,c,C,N) &\quad\Leftrightarrow\\ &\mathrm{Accept}^M(d\,/\,c,C,N)\\ &\vee\ [g(c) = \text{“}\wedge\text{”}\ \wedge\ (\forall c' \in C)((c,c') \in N\ \rightarrow\ \mathrm{Accept}^M(d\,/\,c',C,N))]\\ &\vee\ [g(c) = \text{“}\vee\text{”}\ \wedge\ (\exists c' \in C)((c,c') \in N\ \wedge\ \mathrm{Accept}^M(d\,/\,c',C,N))]\\ \mathrm{Accept}^M(d\,/c) &\quad\Leftrightarrow\quad \mathrm{Accept}^M(d\,/\,c,\mathcal{C}^M(d\,/),\mathcal{NEXT}^M(d\,/))\end{aligned}$$

Now that we have described how accepting configurations of ATMs can be computed in *SRSF*, we turn to the missing bit of initializing the tape with an input word. This initialization part depends on how words are coded in $\mathbb{H}F$, a topic we will discuss next.

2.2 Encoding Words in $\mathbb{H}F$

Any encoding $\nu \colon \Sigma^* \to \mathbb{H}F$ of finite words into $\mathbb{H}F$ gives rise to a class of computable functions over Σ^* which we will denote by SRSF_ν.

Definition 2.4. A function $f \colon \Sigma^* \to \Sigma^*$ is in SRSF_ν, if there exists some $F \in \mathit{SRSF}$ such that the following diagram commutes:

$$\begin{tikzcd}\mathbb{H}F \arrow[r, "F"] & \mathbb{H}F\\ \Sigma^* \arrow[u, "\nu"] \arrow[r, "f"] & \Sigma^* \arrow[u, "\nu"]\end{tikzcd}$$

In general, the function $f\colon (\Sigma^*)^k \to \Sigma^*$ is in SRSF_ν, if

$$\forall w_1, \ldots, w_k \in \Sigma^* \quad \nu(f(w_1, \ldots, w_k)) = F(\nu(w_1), \ldots, \nu(w_k) \,/)$$

for some $F \in \mathit{SRSF}$.

Definition 2.5. We call two encodings ν and ν' equivalent if they can be transformed in each other with functions from *SRSF*. That is, there exist $f, g \in \mathit{SRSF}$ such that

$$\forall w \in \Sigma^* \quad \big(f(\nu(w) \,/) = \nu'(w) \quad \& \quad g(\nu'(w) \,/) = \nu(w)\big)$$

Lemma 2.6. *If ν and ν' are equivalent, then $\mathit{SRSF}_\nu = \mathit{SRSF}_{\nu'}$.*

Several encodings of Σ^* in $\mathbb{H}F$ are possible, but not all will be suitable. We will discuss a few encodings mentioned in the literature.

2.2.1 The Ackerman Encoding

The Ackerman encoding (cf. [?]) $\mathrm{Ack}\colon \mathbb{N} \to \mathbb{H}F$ is given by

$$\mathrm{Ack}(2^{n_1} + 2^{n_2} + \cdots + 2^{n_k}) = \{\mathrm{Ack}(n_1), \mathrm{Ack}(n_2), \ldots, \mathrm{Ack}(n_k)\}$$

for $n_1 > n_2 > \cdots > n_k \geq 0$, $k \geq 0$. This encoding does not give rise to a nice class $\mathit{SRSF}_{\mathrm{Ack}}$ of functions. For example, $\mathit{SRSF}_{\mathrm{Ack}}$ does not include the function $n \mapsto n \dot{-} 1$: Let 2_n denote the exponentiation tower to base 2 of height n, then $\mathrm{Ack}(2_n) = \mathrm{sd}_n$. It is then easy to see that a function F which represents $n \mapsto n \dot{-} 1$ on $\mathbb{H}F$ w.r.t. Ack cannot be dietary, by considering F's behavior on $\mathrm{Ack}(2_n)$.

2.2.2 Two Feasible Encodings

We will now define two feasible encodings ν_{l} and ν_{m}. We call them feasible, because the rank of the encoded word will be of order the length of the word. Actually, both encodings will be equivalent and thus give rise to the same class of functions.

The first encoding, ν_{l}, encodes words as a list using ordered pairs. Let $\nu_{\mathrm{l}}(\lambda) = \emptyset$ and $\nu_{\mathrm{l}}(wx) = (x, \nu_{\mathrm{l}}(w))$, then $\mathrm{rk}(\nu_{\mathrm{l}}(w)) = 2|w| + O(1)$ (the constant term comes from the ranks contributed by elements in Σ.)

The second encoding, ν_{m}, of a word is given as a map from the position of a letter (coded by the rank of a skinny driver) to the letter. Let $x_n \ldots x_1$ denote a word over Σ of length n. We define

$$\nu_{\mathrm{m}}(x_n \ldots x_1) \;=\; \{(\mathrm{sd}_j, x_j)\colon j = 1, \ldots, n\}$$

Then $\mathrm{rk}(\nu_{\mathrm{m}}(w)) = |w| + O(1)$.

We leave it to the reader to verify that ν_{l} and ν_{m} are equivalent.

The main result of this section is the characterization of $\mathit{SRSF}_{\nu_{\mathrm{m}}}$ as the functions computable by an ATM in exponential time with polynomial many alternations.

Theorem 2.7. *A function $f(x)$ is in $\mathit{SRSF}_{\nu_{\mathrm{m}}}$ if, and only if, f can be computed by some machine in $\mathrm{ATIME}(2^{n^c}, n^c)$ for some constant c.*

Here we prove one part of this result, that all functions computable by ATM's in exponential time with polynomial many alternations are in $\mathit{SRSF}_{\nu_{\mathrm{m}}}$. The other part will be the subject of the next section.

Theorem 2.8. *Let f be a function computable in* $\mathrm{ATIME}(2^{n^k}, n^k)$ *for some constant k. Then f is in SRSF_{ν_m}.*

Proof. Let $L \subseteq \Sigma^*$ be a language computable by some $O(2^{n^k})$-time ATM M with $O(n^k)$ many alternations on each computation path. We will define some predicate $P \in \mathit{SRSF}$ such that

$$w \in L \quad \Leftrightarrow \quad P(\nu_{\mathrm{m}}(w))$$

for all $w \in \Sigma^*$. In the following, we will use w to range over words in Σ^*, and m to range over codes of words $\nu_{\mathrm{m}}(\Sigma^*)$.

First, we define a function cwl in *sRud* which computes the length of a coded word as a skinny driver:

$$\mathrm{cwl}(/\, m) = \bigcup_{x \in \bigcup m} h(/\, x, m) \quad h(/\, x, m) = \begin{cases} \{x\} & \text{if } \mathrm{Appl}(m, x) \neq \emptyset \,\&\, \mathrm{Appl}(m, \{x\}) = \emptyset \\ \emptyset & \text{otherwise.} \end{cases}$$
$$\mathrm{cwl}(m\, /) = \mathrm{cwl}(/\, m)$$

Then $\mathrm{cwl}(/\, \nu_{\mathrm{m}}(w)) = \mathrm{sd}_{|w|}$ for $w \in \Sigma^*$.

Second, we have seen that ordinal multiplication is in *SRSF*, so is $f_1(\alpha\, /) = \alpha^k$ for ordinals α. Let

$$f_2(m\, /) = \mathrm{sd}(f_1(\mathrm{rk}(\mathrm{cwl}(m\, /)\, /)\, /)\, /) \ .$$

Fix $w \in \Sigma^*$. Let l denote the ordinal representing $|w|$. We observe that $\mathrm{rk}(\mathrm{sd}_{|w|}\, /) = l$. Thus

$$\begin{aligned} f_2(\nu_{\mathrm{m}}(w)\, /) &= \mathrm{sd}(f_1(\mathrm{rk}(\mathrm{sd}_{|w|}\, /)\, /)\, /) = \mathrm{sd}(f_1(l\, /)\, /) \\ &= \mathrm{sd}(l^k\, /) = \mathrm{sd}_{\mathrm{rk}(l^k)} = \mathrm{sd}_{|w|^k} \end{aligned}$$

Third, we define some functions suitable to produce the initial configuration based on an input word. $\mathrm{null}(\mathrm{sd}_n\, /) = \langle 0, \ldots, 0 \rangle$ points to the first position of the tape:

$$\mathrm{null}(\emptyset\, /) = \emptyset \qquad \mathrm{null}(\{d\}\, /) = (0, \mathrm{null}(d\, /)) \ .$$

$\mathrm{blank}(\mathrm{sd}_n\, /)$ computes the blank tape of length 2^n:

$$\mathrm{blank}(\emptyset\, /) = \sqcup \qquad \mathrm{blank}(\{d\}\, /) = (\mathrm{blank}(d\, /), \mathrm{blank}(d,\, /)) \ .$$

The next function, moveR, computes the movement of the head position to the right. $\mathrm{moveR}(\mathrm{sd}_k, \mathrm{sd}_n\, /\, p)$ computes the head position after moving k steps to the right from position p, assuming that p is of length n:

$$\mathrm{moveR}(\emptyset, e\, /\, p) = p \qquad \mathrm{moveR}(\{d\}, e\, /\, p) = \mathit{Move}_R(e\, /\, \mathrm{moveR}(d, e\, /\, p)) \ .$$

$\mathrm{moveR}(\mathrm{sd}_k, e\, /) = \mathrm{moveR}(\mathrm{sd}_k, e\, /\, \mathrm{null}(e\, /))$ then computes the head position after moving k steps to the right from the first position.

Finally, we can compute initial configurations. $\mathrm{Init}^M(\nu_{\mathrm{m}}(w)\, /)$ computes the initial tape of length $2^{|w|^k}$ with $\nu_{\mathrm{m}}(w)$ standing at the very left end of the tape:

$$\begin{aligned} \mathrm{Init}(m\, /) &= \mathrm{Init}(\mathrm{cwl}(m\, /), f_2(m\, /)\, /\, m) \\ \mathrm{Init}(\emptyset, e\, /\, m) &= \mathrm{blank}(e\, /) \\ \mathrm{Init}(\{d\}, e\, /\, m) &= \mathit{Prt}(e\, /\, \mathrm{moveR}(d, e\, /), \mathrm{Init}(d, e\, /\, m), \mathrm{Appl}(/\, m, d)) \end{aligned}$$

Now we can put things together. Define P as

$$P(m\,/\,) \quad \Leftrightarrow \quad \mathrm{Accept}^M(f_2(m\,/\,)\,/\,\mathrm{Init}^M(\,/\,)) \ .$$

Then $P \in \mathit{SRSF}$ has the desired property that $w \in L$ if and only if $P(\nu_{\mathrm{m}}(w))$ for all $w \in \Sigma^*$. $\square$

2.3 The Converse of Theorem 2.8

For the converse of Theorem 2.8, we shall prove the following theorem.

Theorem 2.9. *Let $f(x)$ be an $\mathit{SRSF}_{\nu_{\mathrm{m}}}$ function. Then f can be computed y some machine in* $\mathrm{ATIME}(2^{n^c}, n^c)$*, for some constant c.*

The proof of Theorem 2.9 will use induction on the formation of $\mathit{SRSF}_{\nu_{\mathrm{m}}}$ functions, with the main induction step being the definition by safe recursion. However, the definition of an $\mathit{SRSF}_{\nu_{\mathrm{m}}}$ function may use intermediate *SRSF* functions which may not be $\mathit{SRSF}_{\nu_{\mathrm{m}}}$ functions. Even worse, these intermediate functions may output sets which have double exponential size $2^{2^{n^c}}$. For instance, the set $\mathcal{C}^M_n$ defined above is an example of a set with double exponential size. For this reason it is necessary to state and prove a generalized form of Theorem 2.9 that will apply to all *SRSF* functions, not just $\mathit{SRSF}_{\nu_{\mathrm{m}}}$ functions.

Definition. A set A has *local cardinality* N provided A and every member of $\mathrm{tc}(A)$ has cardinality $\leq N$.

Definition. An *indexed* tree T is a finite rooted tree in which, for a given node x in T, the children of x are indexed by non-negative integers. That is, for each $i \geq 0$, there is at most one node y which is the child of x of index i. We call y the i-th child of x, however it should be noted that some children may be missing; for example, x might have a third child, but no second child.

Definition. An indexed tree T has *local index rank* N provided that all nodes in T have their children indexed by numbers $< N$.

Definition. Let A be a set with local cardinality N and rank $\leq R$. A can be (non-uniquely) represented by an indexed finite tree T as follows. The subtree of T rooted at the i-th child of the root of T is called the i-th subtree of T. If A is empty, then T is the tree with a single node, namely its root node has no children. For A a general set, T *represents* A is defined by the condition that the elements of A are precisely the sets B for which there is some $i < N$ such that the i-th subtree of T represents B. That is, T represents A provided:

$$A = \{B\colon \text{for some } i, \text{ the } i\text{-th subtree of } T \text{ exists and represents } B\} \ .$$

Definition. Let $\langle i_1, \ldots, i_\ell \rangle$ be a sequence of integers and T be a tree. This sequence denotes a path in T that starts at the root, and proceeds to the i_1-st node of T if it exists, and continues along the path represented by $\langle i_2, \ldots, i_\ell \rangle$ in the i_1-st subtree of T (if it exists). For $I = \langle i_1, \ldots, i_\ell \rangle$, we write T_I for the subtree of T rooted at the end of the path I in T.

Let the *rank of an indexed tree* T be defined by assigning the tree with a single node rank 0, and inductively assigning a general tree rank the supremum of the successors of ranks of children of T's root, i.e. $\max\bigl\{(\text{rank of } T_{\langle i \rangle}) + 1\colon i < N\bigr\}$ where N is the local index rank of

T. We observe that a set of local cardinality N and rank R can be represented by an indexed tree of local index rank N and rank R. Conversely, an indexed tree of local index rank N and rank R represents a set of local cardinality N and rank R.

Definition. An algorithm M *recognizes* a tree T provided that on input $\langle i_1, \ldots, i_\ell \rangle$, M returns a Boolean value indicating whether the path $\langle i_1, \ldots, i_\ell \rangle$ exists in T.

When working with an algorithm M that recognizes a tree T of local index rank N, we shall often have N equal to the value 2^{2^p} for some $p \geq 0$. Note that if the rank of T is bounded by R, then any path $\langle i_1, \ldots, i_\ell \rangle$ is bounded by N^R, and hence is coded by a bit string of length $O(R \log N) = O(R \cdot 2^p)$.

More generally, we may have $N = q^{2^p}$ for some value q, at least for the intermediate parts of some of our proofs.

In our applications, we will have both p and R equal to $n^{O(1)}$, and we usually have $q = 2$. Logarithms are always base 2.

Lemma 2.10. *There are algorithms $M_=$ and $M_\in$ which take as input values $p, R > 0$ and oracles for trees S and T both with local index rank $\leq N = 2^{2^p}$ and rank $\leq R$, and which output Boolean values indicating whether $A = B$ and $A \in B$, respectively, where A and B are the sets represented by S and T, respectively. Furthermore, the algorithms $M_=$ and $M_\in$ run in time $2^p \cdot R^{O(1)}$ using $O(R)$ many alternations.*

Proof. We define slightly more general algorithms $M_=^{S,T}(p, R, I, J)$ and $M_\in^{S,T}(p, R, I, J)$ which decide whether $A_I = B_I$ and $A_I \in B_I$, where A_I and B_J are the sets represented by S_I and T_J.

$M_=^{S,T}(p, R, I, J)$ universally calls two algorithms for checking $A_I \subseteq B_J$ and $A_I \supseteq B_J$. The algorithm for $A_I \subseteq B_J$ first universally chooses $i < N$ and checks whether path $I * \langle i \rangle$ exists in S. If not, it accepts. Otherwise, it then existentially chooses $j < N$, checks that $J * \langle j \rangle$ in T exists and rejects if not. Otherwise, it verifies whether $M_=^{S,T}(p, R, I * \langle i \rangle, J * \langle j \rangle)$. This determines whether $A_I \subseteq B_J$.

The same algorithm is used to determine whether $A_I \supseteq B_J$.

$M_\in^{S,T}(p, R, I, J)$ existentially chooses $j < N$, and checks whether $J * \langle j \rangle$ is in T. If not, it rejects, otherwise it determines whether $M_=^{S,T}(p, R, I, J * \langle j \rangle)$. □

The proof of Lemma 2.10 actually proves a better bound on the number of alternations used by the two algorithms. Namely,

Lemma 2.11. *Lemma 2.10 still holds if the algorithm M is required to use $O(\min\{R_S, R_T\})$ alternations, where R_S and R_T are the ranks of S and T, respectively.*

Proof. The algorithms as described already have alternations bounded in this way. □

Definition. A safe set function $f(\vec{x} \,/\, \vec{a})$ is AEP-computable (where "AEP" stands for "ATIME(Exp, Poly)") provided there are polynomials p, q and r, and an ATM M, such that the following holds. Let $\vec{X}$ and $\vec{A}$ be trees which represent sets $\vec{x}$ and $\vec{a}$. Let the local index rank of $\vec{X}$ and $\vec{A}$ be bounded by N_x and N_a, respectively, and their ranks be bounded by R_x and R_a, respectively. Let $N_{xa} = \max\{N_x, N_a, 2\}$ and $H_a = \max\{R_a, 1\}$. Then $M^{\vec{X},\vec{A}}$ recognizes a tree T which represents the set $f(\vec{x} \,/\, \vec{a})$ such that T has local index rank $\leq N = N_{xa}^{2^{p(R_x)}}$ and rank $\leq R = R_a + r(R_x)$. Furthermore, $M^{\vec{X},\vec{A}}$ runs in time $(H_a \cdot \log N)^{O(1)}$ with $\leq Q = H_a \cdot q(R_x)$ many alternations.

Note that Q depends on R_a multiplicatively, and N depends on only R_x.

Lemma 2.12. *The set equality relation, the set membership relation, the projection functions, the difference function and the pairing function are AEP-computable.*

Proof. For set equality and set membership, use the algorithm from the proof of Lemma 2.10 above. The theorem is obvious for the projection functions since M just computes the same function as one of its oracles. Next consider the pairing function $p(/\, a, b) = \{a, b\}$. If A and B are trees representing the sets a and b, then the tree representing the pair $\{a, b\}$ is

$$\{\langle i\rangle * I \colon I \text{ is a path in } A \text{ if } i = 0, \text{ or a path in } B \text{ if } i = 1\ \} \quad .$$

The property "I is a path in A" (resp, "in B") is computed by invoking one of the oracle inputs. Finally, consider the set difference function $d(/\, a, b) = a \setminus b$. The tree representing the set difference $a \setminus b$ consists of the following paths:

$$\{I = \langle i_1, i_2, ..., i_\ell\rangle : I \text{ is a path in } A, \text{ and for all } j,\ A_{i_1} \text{ is not equal to } B_j\} \quad .$$

M computes this property by universally branching to verify both (a) check that $I \in A$ using the oracle for A, and (b) universally choosing j (this takes $\log N_b$ time where N_b bounds the local index rank of tree B) and invoking $M_=$ to verify that A_{i_1} is not equal to B_j. □

Theorem 2.13. *Every SRSF function is AEP-computable.*

The proof of Theorem 2.13 will show that the formation methods of bounded union, safe composition, and safe recursion preserve the property of being AEP computable. An important ingredient in the construction is how one composes algorithms that use alternation without losing control of the number of alternations. Specifically, suppose that f and g are algorithms that use run times t_f and t_g, and have number of alternations bounded by q_f and q_g. Then, loosely speaking, their composition $f \circ g$ can be computed in time approximately $t_f + t_g$ with $q_f + q_g + O(1)$ many alternations. The basic idea for the algorithm for $f \circ g$ is as follows. Run the algorithm for f; but whenever it needs to query its input (namely, the value of g), it *existentially guesses* the needed input value, and branches universally to both (a) verify the correctness of its guess by executing the algorithm for g, and (b) continue the computation of f. (Alternately, it could branch universally and then existentially.) Note that the algorithm for g is run only once in any given execution path, so contributes only additively to the run time. However, this "basic idea" can increase the number of alternations by the number of times f reads its input (which is more than we can allow); and a better construction is needed. The better construction is as follows:

Algorithm for $f \circ g$: Simulate f by splitting the computation up into existential portions and universal portions. There are at most q_f such portions by assumption. When starting an existential portion, initially guess all input values provided by g that will be needed throughout the computation for this portion. In addition, existentially guess (or, non-deterministically execute) the entire computation for this existential portion using the guessed input values. Then branch universally to either (a) check any one of the guessed input values, by running the algorithm for g and accepting iff it gives the guessed input value, or (b) proceed to the next universal portion. Universal portions of the computation of f are handled dually.

The run time for the algorithm is clearly $O(t_f + t_g)$. And, the number of alternations is at most $q_f + q_g + O(1)$. (The +O(1) is needed for an alternation that may occur as g is invoked; it is also needed to handle the case where f is deterministic and $q_f = 0$.)

Clearly, this construction can be iterated for repeated compositions and this will allow us to handle safe recursion.

Proof of Theorem 2.13. The argument splits into cases of bounded union, safe composition, and safe recursion. The basic idea is to use the method described above for nesting calls to functions, along with the bounds established in the proofs of Theorems 1.5 and 2.3.

Case: Bounded Union. $f(\vec{x} \,/\, \vec{a}, b) = \cup_{z \in b}\, g(\vec{x} \,/\, \vec{a}, z)$. The induction hypothesis that g is AEP-computable gives polynomials p_g, q_g and r_g, and an ATM M_g. Let $\vec{X}$, $\vec{A}$, B be trees representing sets $\vec{x}$, $\vec{a}$, b, with local index ranks bounded by N_x, N_a and N_b, respectively, and ranks bounded by R_x, R_a and R_b, respectively. W.l.o.g. $N_x, N_a, N_b \geq 2$ and $R_a, R_b \geq 1$. Let $N_{xab} = \max\{N_x, N_a, N_b\}$, and $R_{ab} = \max\{R_a, R_b\}$.

We describe the behavior of $M^{\vec{X},\vec{A},B}$ on input $\langle i \rangle * I$: M treats i as a pair (j_1, j_2), and universally (a) checks that $\langle j_1 \rangle$ is a path in B, and (b) runs $M_g^{\vec{X},\vec{A},B_{j_1}}$ on input $\langle j_2 \rangle * I$. Clearly, $M^{\vec{X},\vec{A},B}$ computes a tree T representing $f(\vec{x} \,/\, \vec{a})$. Let N_g be an upper bound to the local index rank of the tree computed by $M_g^{\vec{X},\vec{A},B_{j_1}}$, and R_g an upper bound to its rank. Let Q_g bound the number of alternations for $M_g^{\vec{X},\vec{A},B_{j_1}}$.

T has local index rank bounded by $O(N_g \cdot N_b) = O(N_{xab}^{2^{p_g(R_x)}} \cdot N_b) = N_{xab}^{2^{p_g(R_x)+O(1)}}$ and rank bounded by $R_g \leq R_{ab} + r_g(R_x)$. The algorithms runs in time bounded by

$$(R_{ab} \log N_{xab})^{O(1)} + (R_{ab} \log N_g)^{O(1)} \leq (R_{ab} \log N_{xab}^{2^{p_g(R_x)}})^{O(1)}$$

with $Q_g + 1 \leq R_{ab} \cdot (q_g(R_x) + 1)$ many alternations.

Case: Safe composition. $f(\vec{x} \,/\, \vec{a}) = h(s(\vec{x}/)/t(\vec{x}/\vec{a}))$. Here s and t may be vectors of functions, but we omit this for simplicity (nothing essential is changed in the proof). The induction hypotheses give polynomials p_h, p_s, p_t, q_h, q_s, q_t, r_h, r_s, and r_t, and machines M_h, M_s, and M_t. Let $\vec{X}$ and $\vec{A}$ be trees representing sets $\vec{x}$ and $\vec{a}$, repectively, with local index ranks bounded by N_x and N_a, respectively, and ranks bounded by R_x and R_a respectively. W.l.o.g. $N_x, N_a \geq 2$ and $R_a \geq 1$. Let $N_{xa} = \max(N_x, N_a)$, $N_{st} = \max(N_s, N_t)$, $p_{st} = p_s + p_t$, and $q_{st} = q_s + q_t$. We have that $N_{st} \leq N_{xa}^{2^{p_{st}(R_x)}}$.

Let M be the straightforward algorithm for f, based on composing the algorithms for h, s and t. $M^{\vec{X},\vec{A}}$ will recognize a tree T whose rank is bounded by

$$R_t + r_h(R_s) \leq R_a + r_t(R_x) + r_h(r_s(R_x))$$

so we can choose $r_f = r_t + r_h \circ r_s$. The local index rank of T is bounded by

$$N_{st}^{2^{p_h(R_s)}} \leq \big(N_{xa}^{2^{p_{st}(R_x)}}\big)^{2^{p_h(r_s(R_x))}} = N_{xa}^{2^{p_{st}(R_x)+p_h(r_s(R_x))}} \quad .$$

The run time of M is bounded by, for some $c = O(1)$,

$$\begin{aligned}
&O(\max\{(\mathrm{runtime}(s), \mathrm{runtime}(t)\} + \mathrm{runtime}(h)) \\
&\quad \leq O\Big(\max\{(\log N_x) \cdot 2^{p_s(R_x)}, R_a \cdot (\log N_{xa}) \cdot 2^{p_t(R_x)}\}^c + \big(R_t \cdot (\log N_{st}) \cdot 2^{p_h(R_s)}\big)^c\Big) \\
&\quad \leq O\Big(\big(R_a \cdot (\log N_{xa}) \cdot 2^{p_{st}(R_x)}\big)^c + \big((R_a + r_t(R_x)) \cdot (\log N_{xa}) \cdot 2^{p_{st}(R_x)+p_h(r_s(R_x))}\big)^c\Big) \\
&\quad \leq \big(R_a \cdot (\log N_{xa}) \cdot 2^{p_f(R_x)}\big)^c
\end{aligned}$$

for an appropriately chosen polynomial p_f. Say, $p_f = p_s + p_t + r_t + p_h \circ r_s + O(1)$.

The number of alternations of this algorithm is bounded by

$$\begin{aligned}
&\max\{\text{alternations}(s), \text{alternations}(t)\} + \text{alternations}(h) + O(1)\\
&\quad \leq \max\{q_s(R_x), R_a \cdot q_t(R_x)\} + R_t \cdot q_h(R_s) + O(1)\\
&\quad \leq R_a \cdot q_{st}(R_x) + (R_a + r_t(R_x)) \cdot q_h(r_s(R_x)) + O(1)\\
&\quad \leq R_a \cdot q_f(R_x)
\end{aligned}$$

for an appropriate polynomial q_f.

Case: Safe recursion. $f(x, \vec{y} \,/\, \vec{a}) = h(x, \vec{y} \,/\, \vec{a}, \{f(z, \vec{y} \,/\, \vec{a})\colon z \in x\})$. The induction hypothesis gives polynomials p_h, q_h, r_h, and a machine M_h. Let X, $\vec{Y}$ and $\vec{A}$ be trees representing sets x, $\vec{y}$ and $\vec{a}$, respectively, with local index ranks bounded by N_x, N_y and N_a, respectively, and ranks bounded by R_x, R_y and R_a, respectively. W.l.o.g. $N_x, N_y, N_a \geq 2$ and $R_a \geq 1$. Let $N_{xya} = \max(N_x, N_y, N_a)$, and $R_{xy} = \max(R_x, R_y)$. With M we denote the (yet to be defined) algorithm for computing f.

Let $\bar{f}(x, \vec{y} \,/\, \vec{a})$ be the set $\{f(z, \vec{y} \,/\, \vec{a})\colon z \in x\}$. $\bar{f}$ can be computed by a machine $M_{\bar{f}}^{X,\vec{Y},\vec{A}}$ which on input $\langle i \rangle * I$ first tests whether $\langle i \rangle$ is a path in X, and if so calls $M^{X_i,\vec{Y},\vec{A}}$ on input I. Then, $M^{X,\vec{Y},\vec{A}}$ computes the composition of h with $\bar{f}$ using the above algorithm. Let $R_{\bar{f}}$ ($N_{\bar{f}}$, respectively) denote a bound to the rank (local index rank, respectively) of the tree computed by $M_{\bar{f}}^{X,\vec{Y},\vec{A}}$.

Clearly, $M^{X,\vec{Y},\vec{A}}$ computes a tree T which represents $f(x, \vec{y} \,/\, \vec{a})$. To obtain a bound for the rank of T we can choose r_f similar to the proof of Theorem 1.5: Let $r_f(z) = r'_f(z, z)$ with $r'_f(z, z') = (1 + r_h(z'))(1 + z)$. The same calculation done in that proof carries over here to show by induction on R_x that the rank is $\leq R_a + r'_f(R_x, R_{xy})$.

In order to bound the local index rank of T we choose p_f similar to the proof of Theorem 2.3. Let $p_f(z) = p'_f(z, z)$ for $p'_f(z, z') = (p_h(z') + r_f(z') + O(1)) \cdot (1 + z)$. We show by induction on R_x that the local index rank of T is $\leq N = N_{xya}^{2^{p'_f(R_x, R_{xy})}}$ and that the run time is $\leq (R_a \log N)^{O(1)}$. In case $R_x = 0$ both assertions follow easily. For $R_x > 0$, we calculate as a bound for the local index rank of T

$$\begin{aligned}
\max\{N_{xya}, N_{\bar{f}}\}^{2^{p_h(R_{xy})}} &\leq \max\{N_{xya}, N_x, N_{xya}^{2^{p'_f(R_x - 1, R_{xy})}}\}^{2^{p_h(R_{xy})}}\\
&\leq N_{xya}^{2^{p'_f(R_x - 1, R_{xy}) + p_h(R_{xy})}} \leq N_{xya}^{2^{p'_f(R_x, R_{xy})}}
\end{aligned}$$

The run time of M can be bounded by, for some $c = O(1)$,

$$\begin{aligned}
&O(\text{runtime}(M_h) + \text{runtime}(M_{\bar f})) \\
&\quad \le O\Big(\big(\max\{R_a, R_{\bar f}\} \cdot (\log \max\{N_{xya}, N_{\bar f}\}) \cdot 2^{p_h(R_{xy})}\big)^c \\
&\qquad + \big(R_a \cdot (\log N_{\bar f}) + R_a \cdot (\log N_{xya}) \cdot 2^{p'_f(R_x-1,R_{xy})}\big)^c\Big) \\
&\quad \le O\Big(\big((R_a + r_f(R_{xy})) \cdot (\log N_{xya}) \cdot 2^{p'_f(R_x-1,R_{xy})} \cdot 2^{p_h(R_{xy})}\big)^c \\
&\qquad + \big(R_a \cdot (\log N_{xya}) \cdot 2^{p'_f(R_x-1,R_{xy})} + R_a \cdot (\log N_{xya}) \cdot 2^{p'_f(R_x-1,R_{xy})}\big)^c\Big) \\
&\quad \le \big(\log(N_{xya}) \cdot 2^{p_h(R_{xy})+r_f(R_{xy})+O(1)+p'_f(R_x-1,R_{xy})}\big)^c \\
&\quad = \big(\log(N_{xya}) \cdot 2^{p'_f(R_x,R_{xy})}\big)^c
\end{aligned}$$

Let $q'_f(z,z') = (r_f(z') + O(1)) \cdot q_h(z') \cdot (1+z)$. We will show that the overall number of alterations of $M^{X,\vec{Y},\vec{A}}$ is bounded by $R_a \cdot q'_f(R_x, R_{xy})$ by induction on R_x. Then choosing $q_f(z) = q'_f(z,z)$ gives the desired bound. If $R_x = 0$, the overall number of alterations can be calculated as

$$\text{alternations}(M_h) \le R_a \cdot q_h(R_{xy}) \le R_a \cdot q'_f(0, R_{xy})$$

If $R_x > 0$ we obtain

$$\begin{aligned}
&\text{alternations}(M_h) + \text{alternations}(M_{\bar f}) + O(1) \\
&\qquad \le \max\{R_a, R_{\bar f}\} \cdot q_h(R_{xy}) + R_a \cdot q'_f(R_x - 1, R_{xy}) + O(1) \\
&\qquad \le (R_a + r_f(R_{xy})) \cdot q_h(R_{xy}) + R_a \cdot q'_f(R_x - 1, R_{xy}) + O(1) \\
&\qquad \le R_a \cdot ((r_f(R_{xy}) + O(1)) \cdot q_h(R_{xy}) + q'_f(R_x - 1, R_{xy})) \\
&\qquad = R_a \cdot q'_f(R_x, R_{xy}) \ .
\end{aligned}$$

□

It is easy to verify that Theorem 2.9 is a corollary of Theorem 2.13.

3 Computing on Arbitrary Sets

Our goal in this section is to characterise the safe-recursive functions (i.e., the functions in *SRSF*) in definability-theoretic terms. To achieve this we will use a relativisation of Gödel's L-hierarchy. Our result breaks into two parts: an upper bound result, showing that every safe-recursive function satisfies our definability criterion, and a lower bound result, showing that any function satisfying our definability criterion is in fact safe-recursive. First we introduce:

3.1 The Relativised Gödel Hierarchy

For a transitive set T, define the L^T*-hierarchy* as follows:

$$\begin{aligned}
L^T_0 &= T \\
L^T_{\alpha+1} &= \text{Def}(L^T_\alpha) \\
L^T_\lambda &= \cup_{\alpha<\lambda} L^T_\alpha \quad \text{for limit } \lambda \ ,
\end{aligned}$$

where for any set X, $\mathrm{Def}(X)$ denotes the set of all subsets of X which are first-order definable over the structure $(X, \in)$ with parameters. The following facts are easily verified:

Lemma 3.1. *For any transitive set T:*

1. *T is an element of L_1^T.*
2. *Each L_α^T is transitive and $\alpha \le \beta$ implies $L_\alpha^T \subseteq L_\beta^T$.*
3. *$\mathrm{Ord}(L_\alpha^T) = \mathrm{Ord}(T) + \alpha$, where $\mathrm{Ord}(X)$ denotes $\mathrm{Ord} \cap X$ for any set X.*

Gödel demonstrated the following definability result for the L-hierarchy: For limit α, the sequence $(L_\beta : \beta < \alpha)$ is definable over $(L_\alpha, \in)$ and the definition is independent of α. (See for example [**?**, Chapter II, Lemma 2.8].) His argument readily yields the following refinement, which will be needed for our upper bound result.

Lemma 3.2. *Let $k < \omega$ be sufficiently large, and let T be transitive, α an ordinal and $\varphi(\vec{x}, \vec{y})$ a formula. Let $\mathcal{D}$ consist of all triples $(U, \beta, \vec{p})$ such that for some $\gamma < \alpha$: U is a transitive element of $L_{\gamma+1}^T$, $\gamma + \beta + k < \alpha$ and $\vec{p}$ is a sequence (with the same length as $\vec{y}$) of elements of L_β^U. Then the function with domain $\mathcal{D}$ sending $(U, \beta, \vec{p})$ to $(L_\beta^U, \{\vec{x} \colon L_\beta^U \models \varphi(\vec{x}, \vec{p})\})$ is definable over L_α^T via a definition independent of T, α.*

For our lower bound result we will need the following (see [**?**, Corollary 13.8]):

Lemma 3.3. *(Gödel) There exists a list of functions $G_1(x, y), \ldots, G_{10}(x, y)$ such that for transitive T, $T \cup \bigcup_{1 \le i \le 10} \mathrm{range}(G_i \restriction T \times T)$ is transitive and $\mathrm{Def}(T)$ consists of those subsets of T which belong to the closure of $T \cup \{T\}$ under the G_i's. Moreover, for each i the associated function G_i^* defined by $G_i^*(/\, x, y) = G_i(x, y)$ belongs to sRud.*

3.2 The Upper Bound Result

Recall that we identify finite sequences $\vec{x}$ of sets with individual sets, using Kuratowski pairing. For any set x let $\mathrm{tc}(x)$ denote the transitive closure of x. The rank of $\mathrm{tc}(x)$ (in the von Neumann hierarchy of V_α's) is the same as $\mathrm{rk}(x)$, the rank of x. Given two finite sequences $\vec{x}$, $\vec{y}$, we write $\vec{x} * \vec{y}$ for their concatenation.

Definition 3.4. For sequences $\vec{x}$, $\vec{y}$ and $0 < n \le \omega$ we define $\mathrm{SR}_n(\vec{x} \,/\, \vec{y})$ as $L_{n+\mathrm{rk}(\vec{x})^n}^{\mathrm{tc}(\vec{x}*\vec{y})}$.

Our upper bound result is the following refinement of Theorem 1.5:

Theorem 3.5. *If $f(\vec{x} \,/\, \vec{y})$ is safe-recursive then for some finite n, $f(\vec{x} \,/\, \vec{y})$ is uniformly definable in $\mathrm{SR}_n(\vec{x} \,/\, \vec{y})$, i.e., for some formula $\varphi(\vec{x}, \vec{y}, z)$ we have:*

1. *$f(\vec{x} \,/\, \vec{y})$ belongs to $\mathrm{SR}_n(\vec{x} \,/\, \vec{y})$ for all $\vec{x}$, $\vec{y}$;*
2. *$f(\vec{x} \,/\, \vec{y}) = z$ iff $(\mathrm{SR}_n(\vec{x} \,/\, \vec{y}), \in) \models \varphi(\vec{x}, \vec{y}, z)$.*

To see that this implies Theorem 1.5, note that all elements of $\mathrm{SR}_n(\vec{x} \,/\, \vec{y})$ have rank at most $\mathrm{rk}(\vec{x} * \vec{y}) + n + \mathrm{rk}(\vec{x})^n \le \max(\mathrm{rk}(\vec{x}), \mathrm{rk}(\vec{y})) + k + \mathrm{rk}(\vec{x})^n$ for some finite k, which is bounded by $\max_i \mathrm{rk}(y_i)$ + a polynomial in $\mathrm{rk}(\vec{x})$.

Proof of Theorem 3.5. As in the proof of Theorem 1.5 we proceed by induction on the clauses that generate f as a safe-recursive function. The base cases of *Projection, Difference* and *Pairing* are left to the reader. For *Bounded Union* we have:

$$f(\vec{x} \,/\, \vec{y}, z) = \bigcup_{w \in z} g(\vec{x} \,/\, \vec{y}, w)$$

and by induction there is a finite n such that $g(\vec{x} \,/\, \vec{y}, w)$ is uniformly definable in $\mathrm{SR}_n(\vec{x} \,/\, \vec{y}, w)$. By the definability of union, it then follows from Lemma 3.2 that $f(\vec{x} \,/\, \vec{y}, z)$ is uniformly definable in $\mathrm{SR}_{n+k}(\vec{x} \,/\, \vec{y}, z)$ for sufficiently large k.

Safe Composition

We have:

$$f(\vec{x} \,/\, \vec{y}) = h(\vec{r}(\vec{x} \,/) \,/\, \vec{t}(\vec{x} \,/\, \vec{y}))$$

and by induction n_h, n_{r_i} and n_{t_j} witnessing the theorem for the functions h, r_i for each i and t_j for each j, respectively. By Lemma 3.2 we can choose a large n and combine the uniform definitions of the $r_i(\vec{x} \,/)$'s in the $\mathrm{SR}_{n_{r_i}}(\vec{x} \,/)$'s, of the $t_j(\vec{x} \,/\, \vec{y})$'s in the $\mathrm{SR}_{n_{t_j}}(\vec{x} \,/\, \vec{y})$'s and of $h(\vec{r}(\vec{x} \,/) \,/\, \vec{t}(\vec{x} \,/\, \vec{y}))$ in $\mathrm{SR}_{n_h}(\vec{r}(\vec{x} \,/) \,/\, \vec{t}(\vec{x} \,/\, \vec{y}))$ to produce a uniform definition of $f(\vec{x} \,/\, \vec{y})$ inside $\mathrm{SR}_n(\vec{x} \,/\, \vec{y})$.

Predicative Set Recursion

We have:

$$f(x, \vec{y} \,/\, \vec{z}) = h(x, \vec{y} \,/\, \vec{z}, \{f(w, \vec{y} \,/\, \vec{z}) \colon w \in x\}) \ .$$

Choose $n > 1$ to witness the Theorem for h, i.e., so that $h(x, \vec{y} \,/\, \vec{z}, u)$ is uniformly definable in $\mathrm{SR}_n(x, \vec{y} \,/\, \vec{z}, u)$. Fix $\vec{y}$ and $\vec{z}$. By induction on $\mathrm{rk}(x)$ we show that $f(x, \vec{y} \,/\, \vec{z})$ is uniformly definable in $L^{\mathrm{tc}(\langle x\rangle * \vec{y} * \vec{z})}_{n+\mathrm{rk}(\langle x\rangle * \vec{y})^n \cdot k \cdot (\mathrm{rk}(x)+1)}$ (where $k > n$ is fixed as in Lemma 3.2). If $\mathrm{rk}(x)$ is 0 then we want to show that $f(0, \vec{y} \,/\, \vec{z}) = h(0, \vec{y} \,/\, \vec{z}, 0)$ is an element of $L^{\mathrm{tc}(\langle 0\rangle * \vec{y} * \vec{z})}_{n+\mathrm{rk}(\langle 0\rangle * \vec{y})^n \cdot k}$, which is true by the choice of n. If $\mathrm{rk}(x) > 0$ then by induction we know that for $w \in x$, $f(w, \vec{y} \,/\, \vec{z})$ is uniformly definable in $L^{\mathrm{tc}(\langle w\rangle * \vec{y} * \vec{z})}_{n+\mathrm{rk}(\langle w\rangle * \vec{y})^n \cdot k \cdot (\mathrm{rk}(w)+1)}$; it follows that $\{f(w, \vec{y} \,/\, \vec{z}) \colon w \in x\}$ is uniformly definable over $L^{\mathrm{tc}(\langle x\rangle * \vec{y} * \vec{z})}_{n+\mathrm{rk}(\langle x\rangle * \vec{y})^n \cdot k \cdot \mathrm{rk}(x)}$. By choice of n, $f(x, \vec{y} \,/\, \vec{z}) = h(x, \vec{y} \,/\, \vec{z}, \{f(w, \vec{y} \,/\, \vec{z}) \colon w \in x\})$ is uniformly definable in $L^{\mathrm{tc}(\langle x\rangle * \vec{y} * \vec{z} * \langle \{f(w, \vec{y} \,/\, \vec{z}) \colon w \in x\}\rangle)}_{n+\mathrm{rk}(\langle x\rangle * \vec{y})^n}$ and therefore by Lemma 3.2 also in $L^{\mathrm{tc}(\langle x\rangle * \vec{y} * \vec{z})}_{n+\mathrm{rk}(\langle x\rangle * \vec{y})^n \cdot k \cdot (\mathrm{rk}(x)+1)}$. This completes the induction step.

Now by choosing m large enough so that $n + \mathrm{rk}(\langle x\rangle * \vec{y})^n \cdot k \cdot (\mathrm{rk}(x) + 1)$ is less than $m + \mathrm{rk}(\langle x\rangle * \vec{y})^m$ we have that $f(x, \vec{y} \,/\, \vec{z})$ is uniformly definable in $\mathrm{SR}_m(x, \vec{y} \,/\, \vec{z})$, as desired. $\square$

Note that if there are no safe arguments then $\mathrm{SR}_n(\vec{x} \,/)$ takes a particularly nice form and we have:

Corollary 3.6. *Suppose that $f(\vec{x} \,/)$ is safe-recursive. Then for some finite n and some formula φ we have (for $\vec{x} \neq \langle 0\rangle$):*

1. *$f(\vec{x} \,/)$ belongs to $L^{\mathrm{tc}(\vec{x})}_{n+\mathrm{rk}(\vec{x})^n}$.*

2. *$f(\vec{x} \,/) = y$ iff $L^{\mathrm{tc}(\vec{x})}_{n+\mathrm{rk}(\vec{x})^n} \models \varphi(\vec{x}, y)$.*

For any transitive set T let $\mathrm{SR}(T)$ denote $L^T_{(2+\mathrm{rk}(T))^\omega}$.

Corollary 3.7. *For transitive T, $\mathrm{SR}(T)$ contains $T \cup \{T\}$ and is closed under SRSF functions (i.e., T contains $f(\vec{x} \,/\, \vec{y})$ whenever f is safe-recursive and T contains the components of $\vec{x}$, $\vec{y}$).*

We shall soon see that $\mathrm{SR}(T)$ is in fact the smallest such set.

3.3 The Lower Bound Result

Now we aim for a converse of Theorem 3.5. We begin by showing that a certain initial segment of the L^T-hierarchy can be generated by iteration of a safe-recursive function.

Lemma 3.8. *Suppose that $f(x \,/)$ is safe-recursive with ordinal values and $g(/\, x)$ is safe-recursive with the property that $x \subseteq g(/\, x)$ for all x. By induction on α define $g^\alpha(/\, x)$ by: $g^0(/\, x) = x$, $g^{\alpha+1}(/\, x) = g(/\, g^\alpha(/\, x))$, $g^\lambda(/\, x) = \cup_{\alpha<\lambda} g^\alpha(/\, x)$ for limit λ. Then the function $h(x \,/) = g^{f(x\,/)}(/\, x)$ is safe-recursive.*

Proof. Imitating the proof that multiplication can be defined from addition via a safe recursion, first define the function $k(x, y \,/)$ via a safe recursion as follows:

$$k(x,y\,/) = \begin{cases} y & \text{if } x = 0 \\ g(/ \cup \{k(z,y\,/) \colon z \in x\}) & \text{if } x = \mathrm{Succ}(/ \cup x) \\ \cup \{k(z,y\,/) \colon z \in x\} & \text{otherwise.} \end{cases}$$

Then k is safe-recursive and note that for each ordinal α, $k(\alpha, y \,/) = g^\alpha(/\, y)$. It follows from safe composition that $h(x \,/) = k(f(x \,/), x \,/)$ is also safe-recursive. □

Recall that the rank function $\mathrm{rk}(x \,/)$ is safe-recursive. We say that a function $f(\vec{x} \,/\, \vec{y})$ is *safe-recursive with parameter p* iff for some safe-recursive function $g(\vec{x}, z \,/\, \vec{y})$, we have $f(\vec{x} \,/\, \vec{y}) = g(\vec{x}, p \,/\, \vec{y})$ for all $\vec{x}$, $\vec{y}$.

Corollary 3.9.
1. *The function $\mathrm{tc}(x \,/)$ computing the transitive closure of x, is safe-recursive.*
2. *The function $L(x, T \,/) = L^T_{\mathrm{rk}(x)}$ is safe-recursive with parameter ω.*
3. *For each finite n, the function $\mathrm{SR}_n(\vec{x} \,/)$ is safe-recursive with parameter ω.*

Proof.
1. The transitive closure of x is obtained by iterating the sRud function $g(/\, x) = (x \cup (\cup x))$ $\mathrm{rk}(x)$ times. So the result follows from the previous lemma.
2. The function $g(/\, x) = x \cup$ the union of the ranges of the Gödel functions on x (see Lemma 3.3) belongs to *sRud*. It follows from the previous lemma that the function $g^*(T \,/) = \mathrm{Def}(T) =$ the closure of $T \cup \{T\}$ under g^* (restricted to transitive T) is safe-recursive with parameter ω, as $\mathrm{Def}(T)$ is obtained by iterating g ω times. Similarly, as the function $\mathrm{rk}(x \,/)$ is safe-recursive, an application of the previous lemma gives the safe-recursiveness of $L(x, T \,/)$.
3. This follows from 1 and 2, using the fact that ordinal multiplication is safe-recursive.

□

We therefore get the following partial converse to Theorem 3.5.

Theorem 3.10. *Suppose that for some finite n, $f(\vec{x} \,/\, \vec{y})$ is uniformly definable in $\mathrm{SR}_n(\vec{x} \,/\, \vec{y})$. Then $f(\vec{x} \,/\, \vec{y})$ is safe-recursive with parameter ω. Moreover there is a safe-recursive function $g(\vec{x} \,/\, \vec{y})$ such that $f(\vec{x} \,/\, \vec{y}) = g(\vec{x} \,/\, \vec{y})$ whenever $\vec{x}$ has a component of infinite rank (i.e., whenever $\mathrm{rk}(\vec{x})$ is infinite.)*

Proof. By Corollary 3.9, 3, the function $\mathrm{SR}_n(\vec{x} \,/\, \vec{y})$ is safe-recursive with parameter ω. For any formula $\varphi(\vec{x}, \vec{y}, z)$, the function $g(/\, T, p) = \{(\vec{x}, \vec{y}) : T \vDash \varphi(\vec{x}, \vec{y}, p)\}$ is in *sRud* (see for example [?, Chapter VI, Lemma 1.17]). It follows that any function which is uniformly definable in $\mathrm{SR}_n(\vec{x} \,/\, \vec{y})$ is also safe-recursive with parameter ω. For the "moreover" clause, note that there is a safe-recursive function $f(x\,/)$ whose value is ω for x of infinite rank, and therefore ω can be eliminated as a parameter when $\vec{x}$ has a component of infinite rank. □

Corollary 3.11. *The safe-recursive functions with parameter ω are exactly the functions $f(\vec{x} \,/\, \vec{y})$ which are uniformly definable in $\mathrm{SR}_n(\vec{x} * \langle\omega\rangle \,/\, \vec{y})$ for some finite n.*

Note that the closure of $\{0\}$ under safe-recursive functions is L_ω, the set of hereditarily finite sets and when T is transitive of infinite rank then ω belongs to the safe-recursive closure of T. Therefore we have:

Corollary 3.12. *For transitive T, $\mathrm{SR}(T) = L^T_{(2+\mathrm{rk}(T))^\omega}$ is the smallest set which contains $T \cup \{T\}$ as a subset and is closed under safe-recursive functions.*

We therefore obtain the following hierarchy of iterated safe-recursive closures. Define:

$$\begin{aligned} \mathrm{SR}_0 &= \emptyset \\ \mathrm{SR}_{\alpha+1} &= \mathrm{SR}(\mathrm{SR}_\alpha) \\ \mathrm{SR}_\lambda &= \cup_{\alpha<\lambda} \mathrm{SR}_\alpha \qquad \text{for limit } \lambda. \end{aligned}$$

Corollary 3.13. *For every α, $\mathrm{SR}_{1+\alpha} = L_{\omega^{\omega^\alpha}}$.*

To eliminate the parameter ω from Corollary 3.11 we redefine SR_n slightly, using a slower hierarchy for L^T. Define M^T_α inductively as follows:

$$\begin{aligned} M^T_0 &= T \\ M^T_{\alpha+1} &= M^T_\alpha \cup \bigcup_{1\le i\le 10} \mathrm{range}(G_i \restriction ((M^T_\alpha \cup \{M^T_\alpha\}) \times (M^T_\alpha \cup \{M^T_\alpha\})) \\ M^T_\lambda &= \cup_{\alpha<\lambda} M^T_\alpha \qquad \text{for limit } \lambda. \end{aligned}$$

This hierarchy is very close to Jensen's S-hierarchy, a refinement of his J-hierarchy (see [?, page 244]). We have the following (see [?, page 255]):

Lemma 3.14. *For any transitive set T:*

1. *T is an element of M^T_1.*

2. *Each M^T_α is transitive and $\alpha \le \beta$ implies $M^T_\alpha \subseteq M^T_\beta$.*

3. $\mathrm{Ord}(M^T_\lambda) = \mathrm{Ord}(T) + \lambda$ *for limit* λ.

4. $M^T_\alpha = L^T_\alpha$ *if* α *is* ω *or* $\omega \cdot \alpha = \alpha$. *In particular,* $M^T_{\mathrm{rk}(x)^\omega} = L^T_{\mathrm{rk}(x)^\omega}$ *if* x *has rank greater than* 1.

Definition 3.15. For sequences $\vec{x}$, $\vec{y}$ and $0 < n \leq \omega$ we define $\mathrm{SR}^*_n(\vec{x} \,/\, \vec{y})$ to be $M^{\mathrm{tc}(\vec{x}*\vec{y})}_{n+\mathrm{rk}(\vec{x})^n}$.

Lemma 3.2 and Theorem 3.5 (the upper bound result) go through with L replaced by M and $\mathrm{SR}_n(\vec{x} \,/\, \vec{y})$ replaced by $\mathrm{SR}^*_n(\vec{x} \,/\, \vec{y})$. But now the lower bound result can be improved, as the parameter ω can be dropped in the version of Corollary 3.9 (2), (3) in which L is replaced by M and SR is replaced by SR*: Whereas obtaining $L^T_{\alpha+1}$ from L^T_α requires a safe recursion of length ω, $M^T_{\alpha+1}$ is obtained from M^T_α by a single application of a function in *sRud*. In conclusion, we get the following characterisation:

Theorem 3.16. *The safe-recursive functions are exactly the functions* $f(\vec{x} \,/\, \vec{y})$ *which are uniformly definable in* $\mathrm{SR}^*_n(\vec{x} \,/\, \vec{y})$ *for some finite* n.

3.4 Safe Recursion on Binary ω-Sequences

We let $\{0,1\}^\omega$ denote all ω-sequences of 0's and 1's. Note that if x belongs to $\{0,1\}^\omega$ then x has rank ω. It follows that $\mathrm{SR}_n(x \,/)$ is equal to $L^{\mathrm{tc}(x)}_{\omega^n}$ for $0 < n \leq \omega$. Moreover the latter can be equivalently written as $L_{\omega^n}[x]$, where $L_\alpha[x]$ is the α-th level of the relativised Gödel's L-hierarchy in which x is introduced as a new unary predicate.

Thus the safe-recursive functions restricted to elements of $\{0,1\}^\omega$ as normal inputs take the following form:

$$f(x \,/) = y \quad \text{iff} \quad L_{\omega^n}[x] \vDash \varphi(x, y)$$

for some formula φ.

The following is implicit in the analysis of the "Theory Machine", the universal infinite-time Turing machine considered in [?].

Theorem 3.17. *For any function* $g : \{0,1\}^\omega \to \{0,1\}^\omega$, *the following are equivalent:*

1. *g is computable by an infinite-time Turing machine (see [?]) in time β for some $\beta < \omega^\omega$.*

2. *g is of the form*

$$g(x) = y \quad \textit{iff} \quad L_\beta[x] \vDash \varphi(x, y)$$

for some formula φ and some $\beta < \omega^\omega$.

From this we see that the safe-recursive functions restricted to normal inputs in $\{0,1\}^\omega$ with values in $\{0,1\}^\omega$ are equivalent to the functions computed by an infinite-time Turing machine in time less than ω^ω. Interestingly, these are exactly the functions which are "computable in polynomial time" on an infinite-time Turing machine in the sense of [?].

4 A Machine Model for Safe Recursion

We finish by briefly describing a simple machine model with parallel processors which with the natural bound on running times yields the class of safe-recursive functions.

To each set x assign a processor P_x, which computes in ordinal stages. The value computed by P_x at stage α is denoted by P_x^α. The entire machine M is determined by a function $h(/\, x)$ in sRud and a finite $n > 0$. We write $M = M_h^n$.

P_x^α is defined by induction on α as follows. for any x and α we denote $\{(y, \beta, P_y^\beta)\colon y \in x, \beta \leq \alpha\}$ by $P_{\in x}^{\leq\alpha}$ and $\{(x, \beta, P_x^\beta : \beta < \alpha\}$ by $P_x^{<\alpha}$. Now define:

$$P_x^\alpha = h(/\, P_{\in x}^{\leq\alpha} \cup P_x^{<\alpha}) \ . \tag{4.1}$$

Thus the value computed by processor P_x at stage α is determined by the history of the values of processors P_y for $y \in x$ at stages $\leq \alpha$ together with the values of processor P_x itself at stages $< \alpha$.

The function $f(x\,/) = f^{M_h^n}(x\,/)$ computed by M_h^n is given by: $f(x\,/) = P_x^{\mathrm{rk}(x)^n}$.

Theorem 4.1. *The safe-recursive functions $f(x\,/)$ are exactly those computed by a machine M_h^n for some $h(/\,x)$ in sRud and some finite $n > 0$.*

Proof. It follows from the safe-recursion scheme that the function $g(x, y\,/) = P_x^{\mathrm{rk}(y)^n}$ is safe-recursive (where P_x^α is defined as above, using h). It follows that $f(x\,/) = g(x, x\,/)$, the function computed by M_h^n, is also safe-recursive. Conversely, in view of the improved characterisation of safe-recursive functions given by Theorem 3.16, it suffices to observe that the M-hierarchy, given by applying the Gödel functions iteratively, is obtained by iteration of a function in *sRud* and therefore is captured by Definition (4.1) above. $\square$