The Journal of Symbolic Logic
Volume 80, Number 3, September 2015

SAFE RECURSIVE SET FUNCTIONS

ARNOLD BECKMANN, SAMUEL R. BUSS, AND SY-DAVID FRIEDMAN

Abstract. We introduce the *safe recursive set functions* based on a Bellantoni–Cook style subclass of the primitive recursive set functions. We show that the functions computed by safe recursive set functions under a list encoding of finite strings by hereditarily finite sets are exactly the polynomial growth rate functions computed by alternating exponential time Turing machines with polynomially many alternations. We also show that the functions computed by safe recursive set functions under a more efficient binary tree encoding of finite strings by hereditarily finite sets are exactly the quasipolynomial growth rate functions computed by alternating quasipolynomial time Turing machines with polylogarithmic many alternations.

We characterize 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, relativized to the transitive closure of the function's arguments.

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

§1. Introduction. Polynomial time computation on finite strings is a central notion in complexity theory. Polynomial time in more general settings has been considered by several authors [4, 10]. In this paper, we discuss an attempt to define polynomial time computation on sets in general, based on the Bellantoni–Cook [2] scheme characterizing polynomial time on finite strings in terms of "safe recursion" — we denote our class as *safe recursive set functions* (SRSF). The Bellantoni–Cook approach, which is related to methods implicit in the work of Leivant [14, 15], uses functions with "normal" arguments and "safe" arguments. Functions may be defined by recursion on normal arguments but recursion on safe arguments is not permitted. Our definition of safe recursive set functions uses the same distinction between normal and safe arguments to control the use of set recursion to define functions that act on sets.

Our first results give polynomial upper bounds on the ranks of values computed by general SRSF functions. Then, specializing to the setting of hereditarily finite sets, we establish a double exponential bound on the size of the transitive closures of values computed by SRSF functions. We next establish characterizations of the functions that can be computed by SRSF functions on finite strings encoded by hereditarily finite sets. Namely, using a natural interpretation of finite strings as sets based on lists, we prove that the functions computed by safe recursive set

Received May 15, 2012.
Key words and phrases. Safe recursive, set functions, alternating Turing machines, infinite time Turing machines, polynomial time, rudimentary functions, Jensen hierarchy.

0022-4812/15/8003-0002
DOI:10.1017/jsl.2015.26

functions are exactly those of polynomial growth rate computed by alternating exponential time Turing machine with polynomially many alternations. Using a different, more efficient encoding of finite strings as sets based on binary trees, we prove that the functions computed by safe recursive set functions are exactly those of quasipolynomial growth rate computed by alternating quasipolynomial time Turing machines with polylogarithmically many alternations. Alternating exponential time Turing machines with polynomially many alternations have been considered before, and were shown by Berman [3] to exactly characterize the complexity of validity in the theory of the real numbers as an ordered additive group.

To prove that SRSF functions on encoded words can be computed by alternating Turing machines, we need a much more general result characterizing all SRSF functions on hereditarily finite sets in terms of alternating Turing machines. The characterization is unusual, representing hereditarily finite sets by finite trees, and SRSF functions on hereditarily finite sets as functions recognizing trees by testing whether a given path is present in a tree. We will show in Theorem 3.23 that every such recognizer based on an SRSF function can be computed by some exponential time alternating Turing machine with polynomially many alternations.

Inspired by this part of our work, Arai [1] has subsequently defined the class of *predicatively computable set functions* as a subset of our safe recursive set functions. He replaced one of the basic functions (the rudimentary union scheme, which we denote *bounded union*), which he considered to be impredicative, by some weaker basic functions and schemes (which he denotes *Null, Union, Conditional*$\in$, and *Safe Separation Scheme*.) With this, he obtains, under a natural interpretation of finite strings as sets similar to our list encoding, that the problems decided by predicatively computable set functions are exactly those computed by some polynomial time Turing machine.

Section 4 gives two characterizations of the safe recursive set functions acting on arbitrary sets. The first characterization uses Gödel's L-hierarchy of constructible sets, and the second is in terms of a slower growing hierarchy based on Jensen's S-hierarchy [12]. As a corollary, we prove that the safe recursive set functions on binary ω-sequences are identical to those defined to be computable in "polynomial time" by Schindler [17]. Section 5 gives a parallel machine model for the safe recursive set functions.

We thank the three referees for helpful comments and remarks which helped substantially improve this paper.

§2. Safe recursive set functions.

We consider a subclass of the primitive recursive set functions [13]. As in Bellantoni and Cook's characterization of the polynomial time computable functions [2], we divide arguments 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 semicolon (;) in place of slash (/), and write $f(\vec{x}; \vec{a})$. We use slash instead of semicolon, as we find it improves readability. Set functions whose arguments are typed in this way will be denoted *safe set functions.*

2.1. Safe rudimentary set functions.

We first define safe rudimentary set functions based on rudimentary set functions [12].

https://doi.org/10.1017/jsl.2015.26 Published online by Cambridge University Press

DEFINITION 2.1 (Safe Rudimentary Set Functions). 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 [12]. Let (a,b) denote Kuratowski's ordered pair $\{\{a\},\{a,b\}\}$. The functions $\mathrm{pr_l}$ and $\mathrm{pr_r}$ extract the first and second element from an ordered pair.

- $\mathrm{Union}(/\, a) = \bigcup a$ and $\mathrm{Intersec}(/\, a,b) = a \cap b$ are in SRud, because $\mathrm{Union}(/\, a) = \bigcup_{z \in a} \pi_1^{0,1}(/\, z)$ and $\mathrm{Intersec}(/\, a,b) = c \setminus ((c \setminus a) \cup (c \setminus b))$ for $c = a \cup b = \bigcup\{a,b\}$.
- $\mathrm{Succ}(/\, a) = a \cup \{a\}$, $\mathrm{kop}(/\, a,b) = (a,b)$, $\mathrm{pr_l}(/\, (a,b)) = a$, $\mathrm{pr_r}(/\, (a,b)) = b$ are in SRud:
  $f(/\, c) = \bigcup_{z \in c} \bigcup_{y \in c} (z \setminus y)$ satisfies $f(/\, (a,b)) = \begin{cases} \{b\} & \text{if } a \neq b \\ \emptyset & \text{otherwise,} \end{cases}$
  thus $\mathrm{pr_l}(/\, c) = \bigcup(\bigcup c \setminus f(/\, c))$.
  $g(/\, c) = \bigcup(c \setminus \{\bigcup c\})$ satisfies $g(/\, (a,b)) = \begin{cases} \{a\} & \text{if } a \neq b \\ \emptyset & \text{otherwise,} \end{cases}$
  thus $\mathrm{pr_r}(/\, c) = \bigcup(\bigcup c \setminus g(/\, c))$.
- $\mathrm{Cond}_=(/\, a,b,c,d) = \begin{cases} a & \text{if } c = d \\ b & \text{otherwise} \end{cases}$ is in SRud:
  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 $\mathrm{Cond}_=(/\, a,b,c,d) = g(/\, a,c,d) \cup \bar{g}(/\, b,c,d)$.
- $\mathrm{Cond}_\in(/\, a,b,c,d) = \begin{cases} a & \text{if } c \in d \\ b & \text{otherwise} \end{cases}$ is in SRud:
  Let $h(\,/\, a,c,d) = \bigcup\{g(/\, a,c,z) \colon z \in d\}$ (g as defined for $\mathrm{Cond}_=$),
  and $\bar{h}(\,/\, b,c,d) = b \setminus h(/\, b,c,d)$,

https://doi.org/10.1017/jsl.2015.26 Published online by Cambridge University Press

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 $\mathrm{Cond}_{\in}(/\,a,b,c,d) = h(/\,a,c,d) \cup \bar{h}(/\,b,c,d)$.

- $\mathrm{Appl}(/\,a,b) = \{y\colon (\exists x \in b)(x,y) \in a\}$ is in SRud:

  Let $g(/\,b,c) = \begin{cases} \{\mathrm{pr_r}(/\,c)\} & \text{if } \mathrm{pr_l}(/\,c) \in b \\ \emptyset & \text{otherwise,} \end{cases}$

  then $\mathrm{Appl}(/\,a,b) = \bigcup\{g(/\,b,c)\colon c \in a\}$.
- $\mathrm{Prod}(/\,a,b) = \{(x,y)\colon x \in a, y \in b\} =: a \times b$ is in SRud, by first observing that

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

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

2.2. Predicative set recursion. We extend the safe rudimentary set functions by a predicative set recursion scheme.

Definition 2.2 (Safe Recursive Set Functions). 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.

For a usual function $f(\vec{x})$, that is one where we do not distinguish normal and safe arguments, we say that f is in SRSF if the safe set function $g(\vec{x}\,/)$, given by $g(\vec{x}\,/) := f(\vec{x})$ for all $\vec{x}$, is in SRSF. Since we work exclusively with set functions, we henceforth use the terminology "safe recursive function" instead of "safe recursive set function".

We will show now 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 $1 = \{\emptyset\}$.

- $\mathrm{Add}(x\,/\,a) = \begin{cases} a & \text{if } x = 0 \\ Succ(/\bigcup\{\mathrm{Add}(z\,/\,a)\colon z \in x\}) & \text{if } x = Succ(/\bigcup x) \\ \bigcup\{\mathrm{Add}(z\,/\,a)\colon z \in x\} & \text{otherwise} \end{cases}$

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

  is in SRSF. $\alpha \cdot \beta := \mathrm{Mult}(\beta,\alpha\,/)$ satisfies the usual recursive equations for ordinal multiplication. Observe that for $\alpha \cdot \beta$, both α and β are normal.

https://doi.org/10.1017/jsl.2015.26 Published online by Cambridge University Press

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

The rank of a set x can be defined as $\mathrm{rk}(x) = \bigcup\{\mathrm{rk}(y)+1 \colon y \in x\}$. It is easy to see that $\mathrm{rk}(x\,/\,)$ is in SRSF.

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

DEFINITION 2.3 (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.) Recall 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

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

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

PROPOSITION 2.4 (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

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

- $P(\vec{x}\,/\,\vec{a}) \Leftrightarrow (\exists c \in a_1) Q(\vec{x}\,/\,\vec{a}, c)$ 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:

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

https://doi.org/10.1017/jsl.2015.26 Published online by Cambridge University Press

2.3. Bounding ranks. A very important property of safe recursive functions is that they increase ranks only polynomially. This can be proven similarly to the corresponding Lemma 4.1 in [2]. It should be stressed that the next theorem is *not* restricted to sets of finite rank. In particular, a multivariable *polynomial* $q(\alpha_1, \dots, \alpha_k)$ that takes ordinals as arguments is a sum of terms of the form $\alpha_{i_1}\alpha_{i_2}\cdots\alpha_{i_\ell} n$ where the coefficient $n \in \mathbb{N}$ is positive. This implies that it will be a *monotone* polynomial on ordinals; i.e., if any of its arguments are increased, leaving the other arguments the same, its value does not decrease. We observe that this definition of polynomial is closed under taking sums and products using the associative laws of addition and multiplication, and the distributive law of multiplication. For a list of variables $\vec{x} = x_1, \dots, x_\ell$ we write $\mathrm{rk}(\vec{x})$ instead of $\mathrm{rk}(x_1), \dots, \mathrm{rk}(x_\ell)$.

THEOREM 2.5. *Let f be a function in* SRSF. *There is a polynomial q_f such that*

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

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

Theorem 2.5 will be generalized later by Theorem 4.5.

PROOF. The proof is by induction on the definition of f in SRSF. 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. The case of safe composition, though a straightforward calculation, is essential for maintaining polynomial bounds obtained from the other cases.

If $f(x, \vec{y}\,/\,\vec{a})$ is defined by predicative set recursion from h, then by induction hypothesis we have q_h bounding the rank of h in the above sense. Define q_f as

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

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.

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

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

$$q_f(\gamma, \vec{\beta}) + 1 + q_h(\alpha, \vec{\beta}) \;\leq\; q_f(\alpha, \vec{\beta}). \tag{2.1}$$

Using this we can continue our calculation showing

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

https://doi.org/10.1017/jsl.2015.26 Published online by Cambridge University Press

We finish by proving (2.1):

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

COROLLARY 2.6. *Ordinal exponentiation cannot be computed by a safe recursive function.*

2.4. Safe recursive functions on hereditarily finite sets. In addition to bounding ranks, we can also bound cardinalities for SRSF on hereditarily finite sets. Before doing this, we will consider a special set recursion on hereditarily finite sets which will be useful later. Let HF denote the set of *hereditarily finite sets.*

On HF we will often drive a recursion by some special sets which we denote *skinny drivers*. We define the *skinny driver of rank n,* sd_n, inductively by $\mathrm{sd}_0 = \emptyset$ and $\mathrm{sd}_{n+1} = \{\mathrm{sd}_n\}$. Turning our attention to skinny drivers on HF does not extend the class SRSF, because the function $\mathrm{sd}(x\,/\,) = \mathrm{sd}_{\mathrm{rk}(x)}$ is in SRSF. The latter can be seen by defining $\mathrm{sd}(x\,/\,) = \mathrm{sd}_{\mathrm{rk}(x)}$ in the following way:

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

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

Predicative set recursion based on skinny drivers gives rise to a special set recursion which we call *skinny predicative set recursion.*

PROPOSITION 2.7 (Skinny Predicative Set Recursion). *Let g, h be in* SRSF *of appropriate arities. Then there exists some f in* SRSF *which satisfies*

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

We say that f is defined from g and h by skinny predicative set recursion.

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. ⊣

In the previous subsection, we have seen one important property of SRSF: ranks of sets grow only polynomially under SRSF. Another important property deals with cardinalities of sets, in particular their growth rate on HF under SRSF. Since there are super-exponentially many sets of rank n, Theorem 2.5 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 2.9 will give a substantial improvement over this by showing a double exponential size upper bound. Functions which satisfy such a double

https://doi.org/10.1017/jsl.2015.26 Published online by Cambridge University Press

exponential size upper bound will be called *dietary* – the following definition will make this notion precise.

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

DEFINITION 2.8. A function $f(\vec{x}\,/\,\vec{a})$ in SRSF is called *dietary* if for some polynomial p,

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

for all $\vec{x},\vec{a} \in \mathrm{HF} \setminus \{\emptyset\}$.

Over HF ranks are integers, so p is now an ordinary integer polynomial.

THEOREM 2.9. *All functions in* SRSF *are dietary.*

PROOF. The proof is by induction on the definition of f in SRSF. We will construct 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, and safe composition) are left to the reader. The case of safe composition, though a straightforward calculation employing Theorem 2.5, is essential for maintaining the proposed bounds obtained from the other cases.

If f is defined by predicative set recursion from h, then by induction hypothesis we have that h is dietary, with bounding polynomial q_h. Define $q_f(\alpha,\vec{\beta})$ as $(1+q_h(\alpha,\vec{\beta}))\cdot(1+\alpha)$. We observe that $q_f(\alpha,\vec{\beta}) \geq 1$. We will show that

$$|\,\mathrm{tc}(f(x,\vec{y}\,/\,\vec{a}))| \;\leq\; |\,\mathrm{tc}(\{x,\vec{y},\vec{a}\})|^{2^{q_f(\mathrm{rk}(x),\mathrm{rk}(\vec{y}))}}$$

by $\in$-induction on x. We have

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

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

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

We continue our computation from above:

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

https://doi.org/10.1017/jsl.2015.26 Published online by Cambridge University Press

For the second inequality we use that $q_f(\alpha-1,\vec{\beta}) \geq 1$. The last inequality is derived by:

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

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

The fact that functions in SRSF are dietary is used in Section 3.2.1 to show that certain safe set functions cannot be computed by SRSF.

§3. Computing on hereditarily finite sets. For this section, we restrict our attention to the set HF of hereditarily finite sets. We are interested in complexity classes of alternating time with a bounded number of alternations. An alternating Turing machine [6] is one that is allowed to make both existential and universal moves.

DEFINITION 3.1. Given functions $t(n)$ and $q(n)$, $\mathrm{ATIME}(t(n), q(n))$ is the set of languages which can be decided by some alternating Turing machine which runs, on inputs of length n, in time bounded by $t(n)$, such that the number of alternations on each computation path is bounded by $q(n)$.

Our main result for HF is that under an interpretation of finite strings as sets based on lists, the SRSF functions acting on HF can be characterized in terms of $\mathrm{ATIME}(2^{n^{O(1)}}, n^{O(1)})$; namely, the class of functions of polynomial growth rate computed by alternating exponential time Turing machines with polynomially 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 $\mathrm{ATIME}(2^{n^{O(1)}}, n^{O(1)})$ under polynomial time reductions [3,5,8]. Using a different, more efficient encoding of finite strings as sets based on binary trees, the SRSF functions acting on HF can be characterized as the class of functions of quasipolynomial growth rate computed by alternating quasipolynomial time Turing machines with polylogarithmically many alternations.

In order to state our results precisely, we first have to define function classes based on alternating Turing machines (ATMs), and encodings of binary strings, or more generally words over a finite alphabet, into hereditarily finite sets.

3.1. Functions computed by alternating Turing machines. Let $\mathbb{N}$ denote the set of natural numbers starting from zero, $\mathbb{N} = \{0, 1, 2, \dots\}$. Fix a finite alphabet Σ. With Σ^* we denote the set of words over Σ. Let λ denote the empty word. Given a word $w \in \Sigma^*$, the *length of* w, denoted $|w|$, is the number of symbols forming w.

There are several ways to define what it means for an ATM to compute a function; these are mostly equivalent, at least as long as the function has suitably bounded

https://doi.org/10.1017/jsl.2015.26 Published online by Cambridge University Press

growth rate. We give two equivalent definitions below. Since the ATM can nondeterministically guess its output value as its first computational step, it suffices to define the computation of a function in terms of recognizing the graph of the function.

DEFINITION 3.2. Let Σ be the input/output alphabet of an ATM M. Let $\sqcup$ denote the symbol for a blank tape cell, and assume $\sqcup \notin \Sigma$. Let $f : \Sigma^* \to \Sigma^*$. We say that *$M$ computes the graph of f* if

$$\forall u, v \in \Sigma^* \quad \Big(M \text{ accepts } (u, v) \quad \Leftrightarrow \quad f(u) = v \Big).$$

We say that *M computes the bit-graph of f* if

$$\forall u \in \Sigma^* \, \forall i \in \mathbb{N} \, \forall x \in \Sigma \, \Big(M \text{ accepts } (u, i, x) \;\Leftrightarrow\; i \leq |f(u)| \text{ and the } i\text{-th symbol in } f(u) \text{ is } x, \text{ or } i > |f(u)| \text{ and } x = \text{`}\sqcup\text{'} \Big).$$

For the last part, we assume that i as an argument to M is given in some binary encoding in M's alphabet.

DEFINITION 3.3 (Growth rates). Suppose $r : \mathbb{N} \to \mathbb{N}$ is nondecreasing and $f : \Sigma^* \to \Sigma^*$. We say that *$f$ has growth-rate r* if $|f(w)| \leq r(|w|)$ for all $w \in \Sigma^*$.

We implicitly assume that growth rate functions are time-constructible. The run-time of M will be bounded by a function of only $|u|$. It is possible for M to compute the bit-graph of f even if f has growth rate larger than the run time of M; however, we will not work with functions with such high growth rate.

DEFINITION 3.4 (Functions computed by ATMs). Suppose $t, q : \mathbb{N} \to \mathbb{N}$ and $f : \Sigma^* \to \Sigma^*$. Let f have growth rate t. We say that *f is computed by an ATM M in time $t(|u|)$ with $q(|u|)$ alternations* if $M(u, v)$ runs in time $\leq t(|u|)$ with $\leq q(|u|)$ alternations and computes the graph of f.

For convenience, we henceforth abuse notation by referring to ATM's that run in time $\leq t(|u|)$ with $\leq q(|u|)$ alternations on every input $(u, \cdots)$ as being in $\text{ATIME}(t, q)$. In other words, the time bound t and the alternation bound q are implicitly required to be functions of (only) the length $|u|$ of the first input to M. A function f of growth rate t whose graph is computed by such an M is called an $\text{ATIME}(t, q)$ function.

We could equally well define the $\text{ATIME}(t, u)$ functions in terms of their bit-graph:

THEOREM 3.5. *Let f have growth rate t. If f is in* $\text{ATIME}(t, q)$, *then there is a Turing machine which uses time $O(t(|u|))$ and $\leq q(|u|) + O(1)$ many alternations, and recognizes the bit-graph of f. In the other direction, if the bit-graph of f is recognized by some ATM in time $\leq t(|u|)$ with $\leq q(|u|)$ alternations, then f is in* $\text{ATIME}(O(t(|u|)), q(|u|) + O(1))$.

For this theorem to hold, it is essential that we only consider functions of growth rate at most the run time bound of the machine. The reason for this is that any function computable in the sense of Definition 3.4 based on the graph of the function will satisfy this growth rate bound, as the machine on input (u, v) needs to read

https://doi.org/10.1017/jsl.2015.26 Published online by Cambridge University Press

all of v if it computes the graph of a function: if the machine would accept (u, v) without reading some bit in v, then we could flip this bit obtaining v' and the machine would still accept (u, v') contradicting that a function has to compute a unique value. On the other hand, using the bit-graph version of Definition 3.4, in general functions with larger growth rates are computable, e.g., there are functions of exponential growth rate which are bit-graph computable in polynomial time.

PROOF OF THEOREM 3.5. Let $t, q : \mathbb{N} \to \mathbb{N}$ and $f : \Sigma^* \to \Sigma^*$. For the first direction, let M' be in $\mathrm{ATIME}(t, q)$ computing the graph of f. We construct a machine M computing the bit-graph of f as follows: On input (u, i, x), first existentially guess v, and then verify that the i-th symbol in v is x, and that M' accepts (u, v).

For the other direction, let M' be in $\mathrm{ATIME}(t, q)$ computing the bit-graph of f. We construct a machine M computing the graph of f as follows: On input (u, v), universally verify for $i \leq |v|$ that M' accepts (u, i, x_i) where x_i is the i-th symbol in v, and also that M' accepts $(u, |v| + 1, \sqcup)$. ⊣

3.2. Encoding words in HF. An encoding $v : \Sigma^* \to \mathrm{HF}$ of finite words into HF is any function which is injective. W.l.o.g., we assume that $\Sigma \subset \mathrm{HF}$, and that the elements in Σ do not conflict with our constructions of encodings (the strongest assumption would be that Σ and the image of v are disjoint, but sometimes we want to allow that $v(x) = x$ for $x \in \Sigma$). Any encoding gives rise to a class of computable functions over Σ^* as those which can be represented in the following sense:

DEFINITION 3.6. Let F be a safe set function with normal arguments only, and let $f : \Sigma^* \to \Sigma^*$. We say that F *represents* f *under* v if the following diagram commutes:

$$\begin{array}{ccc} \mathrm{HF} & \xrightarrow{F} & \mathrm{HF} \\ \uparrow{\scriptstyle v} & & \uparrow{\scriptstyle v} \\ \Sigma^* & \xrightarrow{f} & \Sigma^* \end{array}$$

In general, for a function $f : (\Sigma^*)^k \to \Sigma^*$ with k arguments, we say that F *represents* f *under* v if

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

We let SRSF_v denote the set of all functions representable under the encoding v by some function in SRSF.

DEFINITION 3.7. We say that two encodings v and v' are *equivalent* if they can be transformed into each other with functions from SRSF, that is, if there exist $f, g \in \mathrm{SRSF}$ such that

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

The following lemma is an obvious consequence from the definitions.

LEMMA 3.8. *If* v *and* v' *are equivalent, then* $\mathrm{SRSF}_v = \mathrm{SRSF}_{v'}$.

Several encodings of Σ^* in HF are possible, but not all will be suitable in the sense that they lead to function classes with nice properties. We will discuss some encodings mentioned in the literature.

https://doi.org/10.1017/jsl.2015.26 Published online by Cambridge University Press

3.2.1. The Ackermann encoding. The Ackermann encoding $\mathrm{Ack}\colon \mathbb{N} \to \mathrm{HF}$ (cf. [16]) 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 $n_1 > n_2 > \cdots > n_k \geq 0$, $k \geq 0$. It can be extended to binary words $\{0,1\}^*$: First, identify $\mathbb{N}$ in a natural way with the subset of $\{0,1\}^*$ consisting of those binary words which do not start with a leading 0. Then, define $\mathrm{Ack}\colon \{0,1\}^* \to \mathrm{HF}$ in the following way: On $\mathbb{N} \subset \{0,1\}^*$ it has been defined above, and on $\{0,1\}^* \setminus \mathbb{N}$ we set

$$\mathrm{Ack}(0^\ell n) = (\mathrm{Ack}(\ell), \mathrm{Ack}(n)).$$

For $\ell > 0$ and $n \in \mathbb{N}$. This encoding does not give rise to a nice class $\mathrm{SRSF}_{\mathrm{Ack}}$ of functions. For example, $\mathrm{SRSF}_{\mathrm{Ack}}$ does not include a function computing the predecessor function on $\mathbb{N}$, which can be seen as follows. Let 2_n denote iterated exponentiation to base 2 of height n. Then $\mathrm{Ack}(2_n) = \mathrm{sd}_n$. It is then easy to see that any safe set function F which satisfies $F(\mathrm{Ack}(x{+}1)) = \mathrm{Ack}(x)$ for $x \in \mathbb{N}$ cannot be dietary: By considering the behavior on $x = 2_n$, we have $|\,\mathrm{tc}(\mathrm{Ack}(2_n))| = n{+}1$, but $|\,\mathrm{Ack}(2_n - 1)| = 2_{n-1}$.

3.2.2. 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 uses the data structure of *lists* based on ordered pairs to encode words. We denote this encoding with ν_{l} where the subscript "l" stands for "list". We define ν_{l} recursively as follows (recall that λ denotes the empty word, and (a,b) stands for Kuratowski's ordered pair): Let $\nu_{\mathrm{l}}(\lambda) = \emptyset$ and $\nu_{\mathrm{l}}(wx) = (x, \nu_{\mathrm{l}}(w))$. Observe that $\mathrm{rk}(\nu_{\mathrm{l}}(w)) = 2|w| + \Theta(1)$. (The constant term comes from the ranks contributed by elements x in Σ.)

The second encoding uses the concept of a *map* from the position of a letter in a word to the letter. We denote this encoding with ν_{m} where the subscript "m" stands for "map". Let $x_n \cdots x_1$ denote a word over Σ of length n. We define

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

Observe that $\mathrm{rk}(\nu_{\mathrm{m}}(w)) = |w| + \Theta(1)$.

We leave it to the reader to verify that ν_{l} and ν_{m} are equivalent in the sense of Definition 3.7.

3.2.3. An Encoding based on Trees. We now define an encoding which is optimal in the sense that it exhausts the limitations of SRSF in respect to polynomial rank bounds and double exponential size bounds (since SRSF functions are dietary). Binary trees storing letters of words in their leaves have exactly this property, as there are double exponentially many such trees for given rank. We denote this encoding by ν_{t} where the subscript "t" stands for "tree".

Let $x_1 \cdots x_{2^d}$ be a word over $\Sigma \cup \{\sqcup\}$ of length 2^d. We define $T_{x_1 \cdots x_{2^d}} \in \mathrm{HF}$ by recursion on d: Let $T_{x_1} = x_1$, and let

$$T_{x_1 \cdots x_{2^{d+1}}} = (T_{x_1 \cdots x_{2^d}}, T_{x_{2^d+1} \cdots x_{2^{d+1}}}).$$

https://doi.org/10.1017/jsl.2015.26 Published online by Cambridge University Press

Then we define ν_{t} by pairing a skinny driver with a tree, with the skinny driver encoding the height of the tree: Let $\nu_{\mathrm{t}}(\lambda) = (\emptyset, \sqcup)$. If $x_1 \cdots x_n$ denotes a word over Σ of length $n > 0$, let $d = \lceil \log_2(n) \rceil$ and let $x_{n+1} = \cdots = x_{2^d} = \sqcup$, so $x_1 \cdots x_{2^d}$ is $x_1 \cdots x_n$ padded by blanks. Define

$$\nu_{\mathrm{t}}(x_1 \cdots x_n) = (\mathrm{sd}_d, T_{x_1 \cdots x_{2^d}}).$$

Then $\mathrm{rk}(\nu_{\mathrm{t}}(w)) = \Theta(\log(|w|))$ and $|\,\mathrm{tc}(\nu_{\mathrm{t}}(w))| = \Theta(|w|)$.

See Fig. 1 for examples of this encoding.

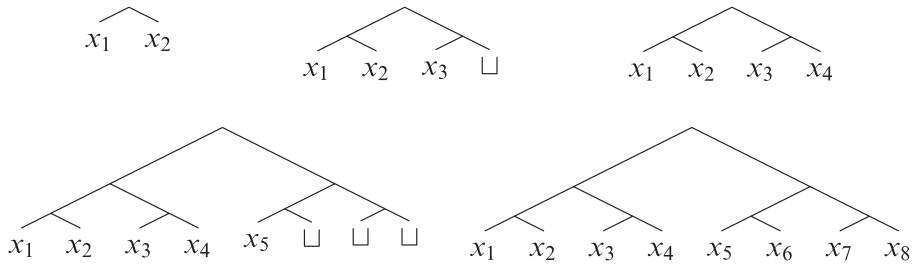

FIGURE 1. Five encodings of words by binary trees: $\nu_{\mathrm{t}}(x_1x_2)$, $\nu_{\mathrm{t}}(x_1x_2x_3)$, $\nu_{\mathrm{t}}(x_1 \cdots x_4)$, $\nu_{\mathrm{t}}(x_1 \cdots x_5)$, and $\nu_{\mathrm{t}}(x_1 \cdots x_8)$.

3.3. Results. We are now able to formulate our main results in this section. The first characterizes $\mathrm{SRSF}_{\nu_{\mathrm{m}}}$ as the polynomial growth-rate functions computable by an ATM in exponential time with polynomial many alternations.

THEOREM 3.9. *A function $f(x)$ is in SRSF_{ν_m} if and only if, for some constant $c > 0$, $|f(x)| = O(|x|^c)$ for all x, and f can be computed by some machine in $\mathrm{ATIME}(2^{n^c}, n^c)$.*

The second characterizes $\mathrm{SRSF}_{\nu_{\mathrm{t}}}$ as the quasi-polynomial growth-rate functions computable by an ATM in quasi-polynomial time with polylogarithmically many alternations.

THEOREM 3.10. *A function $f(x)$ is in SRSF_{ν_t} if and only if, for some constant $c > 0$, f has growth rate $O(2^{\log^c |x|})$, and f is in $\mathrm{ATIME}(2^{\log^c n}, \log^c n)$.*

The proofs for these Theorems will be given in the following two subsections: Theorems 3.11 and 3.12 will provide the "if" directions of Theorems 3.9 and 3.10, respectively. Theorems 3.24 and 3.25 will give the "only if" directions of Theorems 3.9 and 3.10, respectively.

3.4. Simulating alternating Turing machines by safe recursive functions. In order to describe a way of simulating alternating Turing machine computations by safe recursive functions, we will make the definition of an alternating Turing machine, and notations relating to how they compute, more precise. We will consider Turing machines with two one-way tapes, where both heads are always at the same position. A one-way tape is a tape that is one-way infinite to the right and does not restrict the left/right motion of the tape head (except at the left end). We will consider Turing machines which recognize the graph of a function and take two inputs; for technical reasons each input will be stored on a different tape. Formally, an *alternating Turing machine (ATM)* is given by an 8-tuple $(Q, \Sigma, \Gamma, \delta, q_{\mathrm{start}}, q_{\mathrm{accept}}, q_{\mathrm{reject}}, g)$ where the

https://doi.org/10.1017/jsl.2015.26 Published online by Cambridge University Press

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_{start}, the accepting state q_{accept}, and the rejecting state q_{reject}, Σ is the input alphabet, Γ is the work tape alphabet which includes Σ and an additional symbol $\sqcup$ denoting a blank tape cell, and $\delta \subset Q \times \Gamma \times \Gamma \times Q \times \Gamma \times \Gamma \times \{L, R\}$ 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.

We assume that one-way tapes extend infinitely to the right. A *configuration* is given by a quadruple (q, p, u, v) where q is a state in Q, p is a natural number, and u and v are (finite) words over Γ. The triple (q, p, u, v) indicates the machine configuration in which the current state is q, the content of the first tape is u followed by blanks, that of the second tape v followed by blanks, and the heads are positioned on the p-th cell on each tape. The *label* of (q, p, u, v) is given by $g(q)$.

We will not define the behavior of an ATM in full detail, it will be implicit from our discussion. We do use three special conventions, however, that might lead to confusion if not stated explicitly. First, we assume that initially the two input words are written as the first entries from the left on the two one-way tapes, with the heads positioned on the first symbol of each tape. As said before, we assume that the heads are moving in lock-step, thus are always at the same position on each tape. 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. Third, as we are only considering time bounded computation, we will also impose a similar space bound for the tape. This will have no effect on computations in which the head starts from the leftmost tape cell (like in initial configurations), as the space bound will be as big as the time bound.

For the following, we fix an ATM $(Q, \Sigma, \Gamma, \delta, q_{\text{start}}, q_{\text{accept}}, q_{\text{reject}}, g)$, and assume that Γ and Q consist only of sets, and that $\emptyset \notin \Gamma \cup Q$.

Representing configurations. Taking into consideration that functions in SRSF are dietary, and increase ranks of sets only polynomially, we represent configurations as sets in the following way: The content of a single tape 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 then given as $\mathcal{C}_n^M = Q \times \mathcal{P}_n \times \mathcal{T}_n^\Gamma \times \mathcal{T}_n^\Gamma$. All these sets can be defined by functions in SRSF: By skinny predicative set recursion, we can choose HP in SRSF satisfying $\text{HP}(\emptyset\ /) = \{\emptyset\}$ and $\text{HP}(\{d\}\ /) = \text{Prod}(/\ \{0,1\}, \text{HP}(d\ /))$. Then $\text{HP}(\text{sd}_n\ /) = \mathcal{P}_n$. Again by skinny predicative set recursion, we can choose TT^M in SRSF such that $\text{TT}^M(\emptyset\ /) = \Gamma$ and $\text{TT}^M(\{d\}\ /) = \text{Sq}(/\ \text{TT}^M(d\ /))$. Then $\text{TT}^M(\text{sd}_n\ /) = \mathcal{T}_n^\Gamma$. ("TT" stands for "Tree-encoded Tape-contents".) Define $\text{Conf}^M(d\ /)$ as $\text{Prod}(/\ Q, \text{Prod}(/\ \text{HP}(d\ /), \text{Prod}(/\ \text{TT}^M(d\ /), \text{TT}^M(d\ /))))$, then $\text{Conf}^M(\text{sd}_n\ /) = \mathcal{C}_n^M$.

Computing successor configurations. We define a predicate next^M for describing successor configurations according to M: $\text{next}^M(\text{sd}_n\ /\ c, c')$ will be true if $c, c' \in \mathcal{C}_n^M$

https://doi.org/10.1017/jsl.2015.26 Published online by Cambridge University Press

and c' is a possible next configuration from c according to M. It can be defined as a predicate in SRSF in the following way. Here q is the state, p is the tape head position, t_1 and t_2 are the two tape contents encoded as binary trees, and d is a skinny driver.

$$\begin{aligned} next^M(d \,/\, (q,p,t_1,t_2),(q',p',t_1',t_2')) \;\Leftrightarrow\; & \\ \bigvee_{(q,s_1,s_2,q',s_1',s_2',o)\in\delta} \big[& \mathrm{Read}(d \,/\, p,t_1) = s_1 \;\wedge\; \mathrm{Read}(d \,/\, p,t_2) = s_2 \\ & \wedge\; \mathrm{Move}_o(d \,/\, p) = p' \;\wedge\; \mathrm{Prt}(d \,/\, p,t_1,s_1') = t_1' \\ & \wedge\; \mathrm{Prt}(d \,/\, p,t_2,s_2') = t_2' \big]. \end{aligned}$$

Here, $\mathrm{Read}(d \,/\, p,t)$ outputs the symbol at position p in the tape contents t:

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

$\mathrm{Move}_o(d \,/\, p)$ computes the head position obtained by moving from position p in direction $o \in \{L,R\}$, where $\langle 0,\dots,0\rangle$ denotes the leftmost position (see Figure 2):

$$\mathrm{Move}_o(\emptyset \,/\, p) = 0$$

$$\mathrm{Move}_L(\{d\} \,/\, (i,p)) = \begin{cases} \langle 0,\dots,0\rangle & \text{if } (i,p) = \langle 0,\dots,0\rangle\text{; otherwise} \\ (0,p) & \text{if } i = 1 \\ (1,\mathrm{Move}_L(d \,/\, p)) & \text{if } i = 0 \end{cases}$$

$$\mathrm{Move}_R(\{d\} \,/\, (i,p)) = \begin{cases} \langle 1,\dots,1\rangle & \text{if } (i,p) = \langle 1,\dots,1\rangle\text{; otherwise} \\ (1,p) & \text{if } i = 0 \\ (0,Move_R(d \,/\, p)) & \text{if } i = 1 \end{cases}$$

$\langle 0,0,0\rangle$	$\langle 1,0,0\rangle$	$\langle 0,1,0\rangle$	$\langle 1,1,0\rangle$	$\langle 0,0,1\rangle$	$\langle 1,0,1\rangle$	$\langle 0,1,1\rangle$	$\langle 1,1,1\rangle$

FIGURE 2. A tape of length 8 with pointers. Note that the cells are indexed by binary strings in reversed bit order.

$\mathrm{Prt}(d \,/\, p,t,s')$ computes the tape contents obtained by printing the symbol s' at position p in the tape contents t:

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

Our next aim is to define iterations of the successor relation. As the situation of iterating a binary relation R on a set A will occur at various places, we will define this more generally in SRSF. Given two sets r and s (we think of $r \subseteq A \times B$ and $s \subseteq B \times C$) we define their composition $r \circ s$ to be the set $\{(x,z) \in A \times C : (\exists y \in B)(x,y) \in r \wedge (y,z) \in s\}$. This can be defined in SRud as $Comp(/\, r,s) = r \circ s$ because SRud is closed under Boolean connectives and bounded

https://doi.org/10.1017/jsl.2015.26 Published online by Cambridge University Press

quantification. Let A and R be sets (we think of R being a binary relation on A.) We define the iteration of R on A as

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

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

$$\mathrm{Iter}(\emptyset \mathbin{/} R, A) = R \cup \{(x,x)\colon x \in A\}$$
$$\mathrm{Iter}(\{d\} \mathbin{/} R, A) = Comp(\mathbin{/} \mathrm{Iter}(d \mathbin{/} R, A), \mathrm{Iter}(d \mathbin{/} R, A)).$$

Let

$$\mathrm{Next}^{M,g}(d \mathbin{/}) = \left\{(c,c') \in \mathrm{Conf}^M(d \mathbin{/})\colon next^M(d \mathbin{/} c, c') \text{ and } g(c) = g(c')\right\}$$

be the binary relation storing successor configurations according to M for which the label according to g does not change. We define the iteration of $\mathrm{Next}^{M,g}$ by

$$\mathrm{itNext}^{M,g}(d \mathbin{/}) = \mathrm{Iter}(d \mathbin{/} \mathrm{Next}^{M,g}(d \mathbin{/}), \mathrm{Conf}^M(d \mathbin{/})).$$

Let $\mathrm{NEXT}^M(\mathrm{sd}_n \mathbin{/} c, c')$ denote the predicate on configurations $c, c' \in \mathcal{C}_n^M$ which is true if and only if 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. NEXT^M and the binary relation $\mathcal{N}\mathrm{EXT}^M$ based on NEXT^M can be defined as follows:

$$\begin{aligned}\mathrm{NEXT}^M(d \mathbin{/} c, (q', p', t')) \\ \Leftrightarrow\ (\exists c'' \in \mathrm{Conf}^M(d \mathbin{/}))\big[&(c,c'') \in \mathrm{itNext}^{M,g}(d \mathbin{/}) \\ &\wedge\ next^M(d \mathbin{/} c'', (q', p', t')) \\ &\wedge\ [g(c) \neq g(q') \vee q' \in \{q_{\mathrm{accept}}, q_{\mathrm{reject}}\}]\big]\end{aligned}$$

$$\mathcal{N}\mathrm{EXT}^M(d \mathbin{/}) = \left\{(c,c') \in \mathrm{Conf}^M(d \mathbin{/}) \times \mathrm{Conf}^M(d \mathbin{/})\colon \mathrm{NEXT}^M(d \mathbin{/} c, c')\right\}.$$

Computing accepting configurations. 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.) $\mathrm{Accept}^M(\mathrm{sd}_n \mathbin{/} c, C, N)$ will be true if c has an accepting computation of at most n alternations.

$$\mathrm{Accept}^M(\emptyset \mathbin{/} c, C, N) \quad \Leftrightarrow \quad c \in C \wedge state(c) = q_{\mathrm{accept}}$$

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

$$\mathrm{Accept}^M(d \mathbin{/} c) \quad \Leftrightarrow \quad \mathrm{Accept}^M(d \mathbin{/} c, \mathrm{Conf}^M(d \mathbin{/}), \mathcal{N}\mathrm{EXT}^M(d \mathbin{/})).$$

Thus, $\mathrm{Accept}^M(\mathrm{sd}_n \mathbin{/} c)$ will be true if and only if c is a configuration which uses tape trees of height n, such that c has an accepting M-computation whose run time and space is bounded by 2^n, and which uses at most n alternations.

https://doi.org/10.1017/jsl.2015.26 Published online by Cambridge University Press

Preparing initial configurations. The final part for describing which ATM computations can be simulated in SRSF is to compute initial configurations from encoded input words. This will be different for the two encodings under consideration, v_{m} and v_{t}. We start with some safe recursive functions which are used by both.

We start by defining a suitable bounding functions which, depending on the polynomial part in run time and alternation bounds and size parameters of the inputs, provides the height of the tape trees which suffices to capture the whole computation.

Let the polynomial be given by $x^c + c$. Since ordinal addition and multiplication is in SRSF, so is $f_c(\alpha\,/) = \alpha^c + c$ for ordinals α. We define the bounding function by

$$\mathrm{bd}_c(s\,/) = \mathrm{sd}(f_c(\mathrm{rk}(s\,/)\,/)\,/).$$

Then $\mathrm{bd}_c(v_{\mathrm{m}}(w)\,/) = \mathrm{sd}_l$ for some $l \geq |w|^c + c$, and $\mathrm{bd}_c(v_{\mathrm{t}}(w)\,/) = \mathrm{sd}_l$ for some $l \geq \log^c(|w|) + c$.

Next, $\mathrm{null}(\mathrm{sd}_n\,/) = \langle 0, \ldots, 0\rangle$ points to the first position of a tape:

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

The function $\mathrm{blank}(\mathrm{sd}_n\,/)$ computes the blank tape tree of height n (remember, $\sqcup$ denotes the blank tape cell):

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

Representing functions based on v_m. We now turn to encoding initial configurations based on v_{m}. The function, moveR, will be used to compute the head position after some steps 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:

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

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

We compute initial configurations based on v_{m} as follows. "InTT" stands for "Initial Tree-encoded Tape-contents".

$$\mathrm{Init}^M_m(s_1, s_2\,/) = \Big(q_{\mathrm{start}},\ \mathrm{null}(\mathrm{bd}_c(s_1\,/)\,/),\ \mathrm{InTT}^M_m(\mathrm{sd}(s_1\,/), \mathrm{bd}_c(s_1\,/)\,/\,s_1),$$
$$\mathrm{InTT}^M_m(\mathrm{sd}(s_2\,/), \mathrm{bd}_c(s_1\,/)\,/\,s_2)\Big)$$

$$\mathrm{InTT}^M_m(\emptyset, b\,/\,s) = \mathrm{blank}(b\,/)$$

$$\mathrm{InTT}^M_m(\{d\}, b\,/\,s) = \mathrm{Prt}(b\,/\,\mathrm{moveR}(d, b\,/), \mathrm{InTT}^M_m(d, b\,/\,s), \mathrm{Appl}_{\sqcup}(/\,s, \{d\}))$$

$$\mathrm{Appl}_{\sqcup}(/\,s, d) = \begin{cases} \sqcup & \text{if } \mathrm{Appl}(/\,s, \{d\}) = \emptyset \\ \mathrm{Union}(/\ \mathrm{Appl}(/\,s, \{d\})) & \text{otherwise.} \end{cases}$$

Thus, $\mathrm{Init}^M_m(v_{\mathrm{m}}(u), v_{\mathrm{m}}(v)\,/)$ computes the (encoded) initial configuration using tape trees of height $\mathrm{bd}_c(v_{\mathrm{m}}(u)\,/) \geq |u|^c + c$ with u standing at the left end of the first tape tree, and v at the left end of the second tape tree, assuming that $|v| \leq |u|^c + c$.

THEOREM 3.11. *Assume for some constant $c > 0$, that $|f(x)| = O(|x|^c)$ for all x, and that f can be computed by some machine in* $\mathrm{ATIME}(2^{n^c}, n^c)$. *Then f is in* SRSF_{v_m}.

https://doi.org/10.1017/jsl.2015.26 Published online by Cambridge University Press

PROOF. Without loss of generality, we assume that $|f(x)| \leq |x|^c + c$ for all x, and that f is computed by some ATM M computing the graph of f which satisfies our assumptions at the beginning of this subsection, such that the run time of M on input (u, v) is bounded by $2^{|u|^c+c}$ and the number of alternations is bounded by $|u|^c + c$. We will construct some F in SRSF such that $F(v_{\mathrm{m}}(u)\,/\,) = v_{\mathrm{m}}(f(u))$ which proves the claim.

To this end, we first define a suitable superset of all possible outputs of F as a function in SRSF. "mw" stands for "map-encoded words".

$$\begin{aligned}\mathrm{mw}^M(\emptyset\,/\,) &= \{\emptyset\}\\ \mathrm{mw}^M(\{d\}\,/\,) &= \mathrm{mw}^M(d\,/\,) \cup \bigcup_{a\in\Sigma} \left\{s \cup \{(\mathrm{sd}(\{d\}\,/\,), a)\} \colon s \in \mathrm{mw}^M(d\,/\,)\right\}.\end{aligned}$$

Then we define F by

$$F(x\,/\,) = \bigcup \left\{y \in \mathrm{mw}^M(\mathrm{bd}_c(x\,/\,)/\,) \colon \mathrm{Accept}^M(\mathrm{bd}_c(x\,/\,) \,/\, \mathrm{Init}_m^M(x, y\,/\,))\right\}.$$

Given $u \in \Sigma^*$, let $\ell = \mathrm{bd}_c(v_{\mathrm{m}}(u))$, then $\ell \geq |u|^c + c$. We compute

$$\begin{aligned}&F(v_{\mathrm{m}}(u)\,/\,) = s\\ &\quad\Leftrightarrow\quad s \in \mathrm{mw}^M(\ell\,/\,) \text{ and } \mathrm{Accept}^M(\ell \,/\, \mathrm{Init}_m^M(v_{\mathrm{m}}(u), s\,/\,))\\ &\quad\Leftrightarrow\quad \exists v \in \Sigma^{\leq \ell}\ (M \text{ accepts } (u, v) \text{ and } s = v_{\mathrm{m}}(v))\\ &\quad\Leftrightarrow\quad \exists v \in \Sigma^{\leq \ell}\ (f(u) = v \text{ and } s = v_{\mathrm{m}}(v))\\ &\quad\Leftrightarrow\quad s = v_{\mathrm{m}}(f(u)).\end{aligned}$$

⊣

Representing functions based on v_t. We now consider encoding initial configurations based on v_{t}.

$$\begin{aligned}\mathrm{Init}_t^M(s_1, s_2\,/\,) &= \Big(q_{\mathrm{start}},\ \mathrm{null}(\mathrm{bd}_c(s_1\,/\,)/\,),\\ &\qquad \mathrm{InTT}_t^M(\mathrm{bd}_c(s_1\,/\,)\,/\,s_1),\ \mathrm{InTT}_t^M(\mathrm{bd}_c(s_1\,/\,)\,/\,s_2)\Big)\\ \mathrm{InTT}_t^M(\emptyset\,/\,s) &= \begin{cases}\mathrm{pr_r}(/\,s) & \text{if } \mathrm{pr_l}(/\,s) = \emptyset\\ \sqcup & \text{otherwise}\end{cases}\\ \mathrm{InTT}_t^M(\{d\}\,/\,s) &= h(d\,/\,\mathrm{pr_l}(/\,s), \mathrm{pr_r}(/\,s), \mathrm{InTT}_t^M(d\,/\,s))\\ h(d\,/\,d', t, r) &= \begin{cases}t & \text{if } \{d\} = d'\\ (r, \mathrm{blank}(d\,/\,)) & \text{otherwise}.\end{cases}\end{aligned}$$

Thus, $\mathrm{InTT}_t^M(\mathrm{sd}_\ell\,/\,v_{\mathrm{t}}(w))$, for $\ell \geq \log|w|$, will be a tape tree of height ℓ with w standing at the left end. Hence, $\mathrm{Init}_t^M(v_{\mathrm{m}}(u), v_{\mathrm{m}}(v)\,/\,)$ computes the (encoded) initial configuration using tape trees of height $\mathrm{bd}_c(v_{\mathrm{m}}(u)\,/\,) \geq \log^c(|u|) + c$ with u standing at the left end of the first tape, and v at the left end of the second tape, assuming that $|v| \leq 2^{\log^c(|u|)+c}$.

THEOREM 3.12. *Assume for some constant $c > 0$, that $|f(x)| = O(2^{\log^c|x|})$ for all x, and that f can be computed by some machine in* $\mathrm{ATIME}(2^{\log^c n}, \log^c n)$. *Then f is in* SRSF_{v_t}.

https://doi.org/10.1017/jsl.2015.26 Published online by Cambridge University Press

PROOF. Without loss of generality, we assume that $|f(x)| \leq 2^{\log^c |x|+c}$ for all x, and that f is computed by some ATM M computing the graph of f which satisfies our assumptions at the beginning of this subsection, such that the run time of M on input (u, v) is bounded by $2^{\log^c |u|+c}$ and the number of alternations is bounded by $\log^c |u| + c$. We will construct some F in SRSF such that $F(\nu_{\mathrm{t}}(u)\,/) = \nu_{\mathrm{t}}(f(u))$ which proves the claim.

To this end, we first define a suitable superset of all possible outputs of F as a function in SRSF, that is, a set containing $\nu_{\mathrm{t}}(w)$ for $w \in \Sigma^*$ in some "unique" way. "tw" stands for "tree-encoded words".

$$\mathrm{tw}^M(\emptyset\,/) = \{\emptyset\} \times \mathrm{TT}^M(\emptyset\,/)$$
$$\mathrm{tw}^M(\{d\}\,/) = \mathrm{tw}^M(d\,/) \cup \left\{(\{d\}, (u, v))\colon (u, v) \in \mathrm{TT}^M(\{d\}\,/) \wedge v \neq \mathrm{blank}(d\,/)\right\}.$$

Then we define F by

$$F(x\,/) = \bigcup \left\{y \in \mathrm{tw}^M(\mathrm{bd}_c(x\,/)\,/)\colon\ \mathrm{Accept}^M(\mathrm{bd}_c(x\,/)\,/\,\mathrm{Init}_t^M(x, y\,/))\right\}.$$

Given $u \in \Sigma^*$, let $\ell = \mathrm{bd}_c(\nu_{\mathrm{t}}(u))$, then $\ell \geq \log^c |u| + c$. We compute

$$\begin{aligned}
&F(\nu_{\mathrm{t}}(u)\,/) = s\\
&\quad\Leftrightarrow\quad s \in \mathrm{tw}^M(\ell\,/) \text{ and } \mathrm{Accept}^M(\ell\,/\,\mathrm{Init}_t^M(\nu_{\mathrm{t}}(u), s\,/))\\
&\quad\Leftrightarrow\quad \exists v \in \Sigma^{\leq \ell}\ (M \text{ accepts } (u, v) \text{ and } s = \nu_{\mathrm{t}}(v))\\
&\quad\Leftrightarrow\quad \exists v \in \Sigma^{\leq \ell}\ (f(u) = v \text{ and } s = \nu_{\mathrm{t}}(v))\\
&\quad\Leftrightarrow\quad s = \nu_{\mathrm{t}}(f(u)).
\end{aligned}$$

⊣

3.5. Simulating safe recursive functions by alternating Turing machines. The aim of this subsection is to prove the converses of Theorems 3.11 and 3.12. For example, we want to show that all functions in $\mathrm{SRSF}_{\nu_{\mathrm{m}}}$ can be computed by some machine in $\mathrm{ATIME}(2^{n^c}, n^c)$, for some constant c. The proof of this result will use induction on the formation of $\mathrm{SRSF}_{\nu_{\mathrm{m}}}$ functions, with the main induction step being the definition by predicative set recursion. However, the definition of an $\mathrm{SRSF}_{\nu_{\mathrm{m}}}$ function may use intermediate SRSF functions which may not be $\mathrm{SRSF}_{\nu_{\mathrm{m}}}$ functions. Even worse, these intermediate functions may output sets which have double exponential size $2^{2^{n^c}}$. For instance, the SRSF function $\mathrm{Conf}^M(\mathrm{sd}_n\,/)$ defined above computes a set of double exponential size. For this reason, it is necessary to state and prove a generalized form of the above assertion that will apply to all SRSF functions, not just $\mathrm{SRSF}_{\nu_{\mathrm{m}}}$ functions. This will be done by introducing a notion of "AEP-computability" based on alternating Turing machines which recognize sets via a tree representation, culminating in Theorem 3.23 showing that every SRSF function is AEP-computable.

DEFINITION 3.13. A set A has *local cardinality* N provided N is the smallest number such that A and each member of $\mathrm{tc}(A)$ has cardinality $\leq N$.

DEFINITION 3.14. 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 nonnegative 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

https://doi.org/10.1017/jsl.2015.26 Published online by Cambridge University Press

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 3.15. An indexed tree T has *local index size* N provided N is the smallest number such that all nodes in T have their children indexed by numbers $< N$.

DEFINITION 3.16. Let A be a set with local cardinality N. A can be (nonuniquely) represented by an indexed finite tree T of local index size N 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. 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 3.17. Let $\langle i_1, \ldots, i_\ell\rangle$ be a sequence of integers and T be a tree. We say that path $\langle i_1, \ldots, i_\ell\rangle$ exists in T if and only if either $\ell = 0$, or $\ell > 0$ the i_1-st node of T exists and the path $\langle i_2, \ldots, i_\ell\rangle$ exists in the i_1-st subtree of T. If $I = \langle i_1, \ldots, i_\ell\rangle$ exists in T, we write T_I for the subtree of T rooted at the end of the path I in T. If I does not exist in T, we let T_I be undefined.

The *rank of an indexed tree* T is 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.,

$$\max\left\{(\text{rank of } T_{\langle i\rangle}) + 1\colon i\text{-th node in } T \text{ exists}\right\}.$$

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

DEFINITION 3.18. 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 size 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 $\langle i_1, \ldots, i_\ell\rangle$ in T will have $\ell \leq R$ and $i_j \leq N$ for $j = 1, \ldots, \ell$, 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 $n^{O(1)}$, where n will denote the size of the input, and we usually have $q = 2$. Logarithms are always base 2.

LEMMA 3.19. *There are algorithms $M_=$ and $M_\in$ which take as input value $p > 0$ and oracles recognizing trees S and T both with local index size $\leq N = 2^{2^p}$ 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.*

https://doi.org/10.1017/jsl.2015.26 Published online by Cambridge University Press

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

$M_{=}^{S,T}(p,I,J)$ 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 $M_{=}^{S,T}(p, I * \langle i \rangle, J * \langle j \rangle)$. This determines whether $A_I \subseteq B_J$. The same algorithm is used to determine whether $A_I \supseteq B_J$.

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

The proof of Lemma 3.19 actually proves a better bound on the number of alternations used by the two algorithms. Namely,

LEMMA 3.20. *Lemma* 3.19 *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 3.21. A safe set function $f(\vec{x} \,/\, \vec{a})$ is said to be *AEP-computable* (where "AEP" stands for "ATIME(Exp, Poly)") provided there are polynomials p, q and r, and an oracle 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 size 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. W.l.o.g., $R_a \geq 1$. Let $N_{xa} = \max\{N_x, N_a, 2\}$. 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 size $\leq N = N_{xa}^{2^{p(R_x)}}$ and rank $\leq R = R_a + r(R_x)$. Furthermore, $M^{\vec{X},\vec{A}}$ runs in time $(R_a \cdot \log N)^{O(1)}$ with $\leq Q = R_a \cdot q(R_x)$ many alternations.

Note that Q depends on R_a multiplicatively, and N depends on R_x but not on R_a. The runtime bound depends polynomially on R_a and exponentially on R_x (via N). Without making this explicit, we tacitly assume that some information on local index sizes of input trees is passed on to an AEP-algorithm, e.g. in form of a bound on $\log\log N_{xa}$. From this the algorithm can compute bounds on local index sizes of any intermediate tree it needs in the computation. For example, when computing the composition of two AEP-algorithms, it needs to compute information on the local index size of the intermediate tree computed by the first AEP-algorithm, and pass it on to the second algorithm for computing the final tree. Without passing information about local index sizes to AEP-algorithms, testing for the empty set for example would not be (AEP-)computable (only "semi-computable").

LEMMA 3.22. *The set equality relation on safe arguments, the set membership relation on safe arguments, the projection functions* $\pi_j^{n,m}(\vec{x} \,/\, \vec{a})$, *the difference function* $\mathrm{d}(/\, a, b)$, *and the pairing function* $\mathrm{p}(/\, a, b)$ *are AEP-computable.*

PROOF. For set equality and set membership, use the algorithm from the proof of Lemma 3.19 above. The lemma is obvious for the projection functions since M just computes the same function as one of its oracles. Next consider the pairing function

https://doi.org/10.1017/jsl.2015.26 Published online by Cambridge University Press

$p(/\, a, b) = \{a, b\}$. If A and B are trees representing the sets a and b, then the tree representing the pair $\{a, b\}$ is

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

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:

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

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 size of tree B) and invoking $M_=$ to verify that $A_{\langle i_1\rangle}$ is not equal to $B_{\langle j\rangle}$. ⊣

We are now able to state and prove a general characterization of how SRSF functions can be computed by alternating Turing machines. This will be applied later in Theorems 3.24 and 3.25 to obtain the specific characterizations that a function in SRSF_{v_m} can be computed by $\mathrm{ATIME}(2^{n^c}, n^c)$, for some c, and that a function in SRSF_{v_t} can be computed by $\mathrm{ATIME}(2^{\log^c n}, \log^c n)$, for some c.

THEOREM 3.23. *Every* SRSF *function is AEP-computable.*

The proof of Theorem 3.23 will show that the formation methods of bounded union, safe composition, and predicative set 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 to simulate the algorithm for f as follows. The algorithm for f makes queries to its input; namely, it queries whether a path $\langle i_1, \ldots, i_\ell\rangle$ exists in the tree representing its input. Each such query during the execution of f is handled by first *existentially guessing* the needed (Boolean) answer to the query and then *branching universally* to both (a) verify the correctness of the guessed answer by executing the algorithm for g, and (b) continue the computation of f. (Alternately, it could branch first universally and then existentially.) Note that the algorithm for g is run only once in any given execution path, and thus 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 described next.

Algorithm for $f \circ g$: Let M_f be the algorithm for f which recognizes the tree representing the set computed by f; let M_g be a similar algorithm for g. Simulate M_f by splitting the computation up into existential portions and universal portions. There are at most q_f many such portions by assumption. When starting the simulation of an existential portion, initially guess the entire computation for this existential portion including guessing all answers to queries asked to M_g during this part of the computation. Check that the guessed computation represents a valid computation, modulo the correctness of the guessed answers to the queries to M_g,

https://doi.org/10.1017/jsl.2015.26 Published online by Cambridge University Press

and (temporarily) write the entire computation for this existential portion to a tape. Then branch universally to both (a) universally select one of the guessed answers to the queries to M_g, and check the correctness of that query by running M_g and accept if it confirms the guessed answer and otherwise reject; and (b) proceed to the next, universal portion of the computation of M_f. For (a), it is important that the guessed computation for the existential portion has been written to tape (this is possible since there is no separate space bound on the computation of f), as this means that all queries to M_g and their guessed answers are available to be selected for verification of correctness.

Universal portions of the computation of M_f are handled dually to existential portions.

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. This will allow us to handle predicative set recursion.

Proof of Theorem 3.23. The argument splits into cases of bounded union, safe composition, and predicative set 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 2.5 and 2.9.

Case: Bounded Union. $f(\vec{x} \,/\, \vec{a}, b) = \bigcup_{z \in b} g(\vec{x} \,/\, \vec{a}, z)$. 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 sizes 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 $M_g^{\vec{X},\vec{A},B_{\langle j_1 \rangle}}$ on input $\langle j_2 \rangle * I$. By construction, $M^{\vec{X},\vec{A},B}$ computes a tree T representing $f(\vec{x} \,/\, \vec{a}, b)$. Let N_g be an upper bound to the local index size of the tree computed by $M_g^{\vec{X},\vec{A},B_{\langle j_1 \rangle}}$, and R_g an upper bound to its rank. Let Q_g bound the number of alternations for $M_g^{\vec{X},\vec{A},B_{\langle j_1 \rangle}}$.

T has local index size 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 first part of the algorithm, which decomposes the first entry i of the input into two parts of a pair j_1 and j_2, runs in time $|i|^{O(1)}$ which can be bounded by $(\log N_{xab}^{2^{p_g(R_x)+O(1)}})^{O(1)} = (\log N_{xab}^{2^{p_g(R_x)}})^{O(1)}$. Thus, overall the algorithm runs in time bounded by

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

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

https://doi.org/10.1017/jsl.2015.26 Published online by Cambridge University Press

$\vec{x}$ and $\vec{a}$ with local index sizes bounded by N_x and N_a, 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 algorithm for f, based on composing the algorithms for h, s and t using the above-described algorithm for composition. $M^{\vec{X},\vec{A}}$ 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 size of T is bounded by

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

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

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

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: Predicative set 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 sizes 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 $\{f(z, \vec{y} \,/\, \vec{a}) \colon z \in x\}$. This set can be recognized 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 $M^{X_{\langle i \rangle},\vec{Y},\vec{A}}$ on input I. Then, $M^{X,\vec{Y},\vec{A}}$ computes the composition of h with $\bar{f}$ using the earlier-described algorithm. Let $R_{\bar{f}}$ ($N_{\bar{f}}$, respectively) denote a bound to the rank (local index size, 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 2.5: Let

https://doi.org/10.1017/jsl.2015.26 Published online by Cambridge University Press

$r_f(z) = r'_f(z,z)$ 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 of T is $\leq R_a + r'_f(R_x, R_{xy})$.

In order to bound the local index size of T we choose p_f similar to the proof of Theorem 2.9. 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 size of T is bounded by N defined as

$$N_{xya}^{2^{p'_f(R_x,R_{xy})}}$$

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 size 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 $c = O(1)$,

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

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

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

If $R_x > 0$ we obtain

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

⊣

We are now ready to prove the converses of Theorems 3.11 and 3.12, and thus finishing the proofs of Theorems 3.9 and 3.10, respectively.

https://doi.org/10.1017/jsl.2015.26 Published online by Cambridge University Press

THEOREM 3.24. *Assume f is in* SRSF_{ν_m}. *Then there is some constant $c > 0$ such that $|f(x)| = O(|x|^c)$ for all x, and that f can be computed by some machine in* $\mathrm{ATIME}(2^{n^c}, n^c)$.

PROOF. Assume $f(x)$ is in SRSF_{ν_m}, then there is some F in SRSF such that $F(\nu_m(u)/) = \nu_m(f(u))$ for all $u \in \Sigma^*$. Using Theorem 2.5 together with $\mathrm{rk}(\nu_m(u)) = \Theta(|u|)$, we obtain some polynomial q such that $|f(u)| \leq q(|u|)$.

Let $u \in \Sigma^*$. $\nu_m(u)$ can be represented by some canonical tree $T_{\nu_m(u)}$ of local index size $O(|u|)$ and rank $O(|u|)$. Furthermore, there is a (deterministic) Turing machine M_{ν_m} taking two inputs such that, if the first input is fixed to u, M_{ν_m} acts as a recognizer for $T_{\nu_m(u)}$ and runs in time $O(|u|)$ (independent of the second input).

By Theorem 3.23, F is AEP-computable. Thus, there is some ATM M^X and polynomial p such that $M^{T_{\nu_m(u)}}$ recognizes a tree T representing $F(\nu_m(u)/)$, the tree T has local index size $\leq 2^{p(|u|)}$ and rank $\leq p(|u|)$, and $M^{T_{\nu_m(u)}}$ runs in time $\leq 2^{p(|u|)}$ with $\leq p(|u|)$ many alternations. Using $M_{\nu_m}(u)$ to answer queries to $T_{\nu_m(u)}$, we obtain a machine M which takes two inputs such that, if the first input is fixed to u, M acts as a recognizer for T and runs in time $\leq 2^{p(|u|)+1}$ with $\leq p(|u|)$ many alternations.

We define a machine $\overline{M}$ deciding the graph of f as follows: On inputs $u, v \in \Sigma^*$, $\overline{M}$ first verifies that $|v| \leq q(|u|)$ and rejects if this is not the case. Then, $\overline{M}$ runs $M_{=}^{T, T_{\nu_m(v)}}$, using $M(u)$ for recognizing T, and $M_{\nu_m}(v)$ for recognizing $T_{\nu_m(v)}$. By Lemma 3.19, this algorithm runs in time $\leq 2^{p(|u|)+1} \cdot p(|v|)^{O(1)}$ with $O(p(|u|) + |v|)$ many alternations. ⊣

THEOREM 3.25. *Assume f is in* SRSF_{ν_t}. *Then there is some constant $c > 0$ such that $|f(x)| = O(2^{\log^c |x|})$ for all x, and that f can be computed by some machine in* $\mathrm{ATIME}(2^{\log^c n}, \log^c n)$.

PROOF. The proof is similar to the previous one. Assume $f(x)$ is in SRSF_{ν_t}, then there is some F in SRSF such that $F(\nu_t(u)/) = \nu_t(f(u))$ for all $u \in \Sigma^*$. Using Theorem 2.5 together with $\mathrm{rk}(\nu_t(u)) = \Theta(\log(|u|))$, we obtain some polynomial q such that $|f(u)| \leq 2^{q(\log|u|)}$.

Let $u \in \Sigma^*$. $\nu_t(u)$ can be represented by some canonical tree $T_{\nu_t(u)}$ of local index size $O(1)$ and rank $O(\log|u|)$. Furthermore, there is a (deterministic) Turing machine M_{ν_t} taking two inputs, such that if the first is set to u, acts as a recognizer for this tree, and runs in time $O(|u|)$ (independent of the second input).

The constructions of machines are now the same as in the previous proof, with the difference that M_{ν_m} is replaced with M_{ν_t}. Only the resulting bounds on running time and alternations are changed due to the different input encoding. The tree T representing the value of f will have local index size $\leq 2^{p(\log|u|)}$ and rank $\leq p(\log|u|)$ for some polynomial p, and the machine M which on input u acts as a recognizer for T can be constructed with running time $\leq 2^{p(\log|u|)}$ using $\leq p(\log|u|)$ many alternations. We then obtain a machine deciding the graph of f with the required bounds on run time and alternations as before. ⊣

§4. Computing on arbitrary sets. Our goal in this section is to characterize the safe recursive functions (i.e., the functions in SRSF) in definability-theoretic terms. To achieve this we will use a relativization of Gödel's L-hierarchy. Our result breaks into two parts: an upper bound result, showing that every safe recursive function satisfies

https://doi.org/10.1017/jsl.2015.26 Published online by Cambridge University Press

our definability criterion, and a lower bound result, showing that any function satisfying our definability criterion is in fact safe recursive. First we introduce:

4.1. The relativized Gödel hierarchy. For a transitive set T, define the L^T-*hierarchy* as follows:

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

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 4.1. *For any transitive set T:*

1. *T is an element of L^T_1.*
2. *Each L^T_α is transitive and $\alpha \leq \beta$ implies $L^T_\alpha \subseteq L^T_\beta$.*
3. *$\mathrm{Ord}(L^T_\alpha) = \mathrm{Ord}(T) + \alpha$, 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 [7, Chapter II, Lemma 2.8].) His argument readily yields the following refinement, which will be needed for our upper bound result.

LEMMA 4.2. *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 γ, U is a transitive element of $L^T_{\gamma+1}$, $\gamma + k \cdot \beta < \alpha$, and $\vec{p}$ is a sequence (with the same length as $\vec{y}$) of elements of L^U_β. Then the function with domain $\mathcal{D}$ sending $(U, \beta, \vec{p})$ to $(L^U_\beta, \{\vec{x} \colon L^U_\beta \vDash \varphi(\vec{x}, \vec{p})\})$ is definable over L^T_α via a definition independent of T, α.*

For our lower bound result we will need the following (see [11, Corollary 13.8]):

LEMMA 4.3 (Gödel). *There exists a list of functions $G_1(x, y), \ldots, G_{10}(x, y)$ such that for transitive T, $\bigcup_{1 \leq i \leq 10} \mathrm{range}(G_i \restriction T \times T)$ is a transitive set containing T as a subset 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.

4.2. 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 4.4. For sequences $\vec{x}$, $\vec{y}$ and $0 < n \leq \omega$ we define $\mathrm{SR}_n(\vec{x}\,/\,\vec{y})$ as $L^{\mathrm{tc}(\vec{x}*\vec{y})}_{(2+\mathrm{rk}(\vec{x}))^n}$.

Our upper bound result is the following refinement of Theorem 2.5:

THEOREM 4.5. *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. *$f(\vec{x}\,/\,\vec{y}) = z$ if and only if $(\mathrm{SR}_n(\vec{x}\,/\,\vec{y}), \in) \vDash \varphi(\vec{x}, \vec{y}, z)$.*

https://doi.org/10.1017/jsl.2015.26 Published online by Cambridge University Press

To see that this implies Theorem 2.5, note that all elements of $\mathrm{SR}_n(\vec{x}\,/\,\vec{y})$ have rank at most $\mathrm{rk}(\vec{x}*\vec{y})+(2+\mathrm{rk}(\vec{x}))^n \le \max(\mathrm{rk}(\vec{x}),\mathrm{rk}(\vec{y}))+k+(2+\mathrm{rk}(\vec{x}))^n$ for some finite k, which is bounded by $\max_i \mathrm{rk}(y_i)$ + a polynomial in $\mathrm{rk}(\vec{x})$.

Proof of Theorem 4.5. As in the proof of Theorem 2.5 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 4.2 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. Suppose

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

The induction hypothesis gives values 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 4.2 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. Suppose

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

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 $\vec{z}$. We first show by induction on $\mathrm{rk}(x)$ that $f(x,\vec{y}\,/\,\vec{z})$ is an element of $L^{\mathrm{tc}(\langle x\rangle*\vec{y}*\vec{z})}_{k\cdot(2+\mathrm{rk}(\langle x\rangle*\vec{y}))^n\cdot(\mathrm{rk}(x)+1)}$ (where $k>n$ is fixed as in Lemma 4.2). 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 $L^{\mathrm{tc}(\langle 0\rangle*\vec{y}*\vec{z})}_{k\cdot(2+\mathrm{rk}(\langle 0\rangle*\vec{y}))^n}$, 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 in $L^{\mathrm{tc}(\langle w\rangle*\vec{y}*\vec{z})}_{k\cdot(2+\mathrm{rk}(\langle w\rangle*\vec{y}))^n\cdot(\mathrm{rk}(w)+1)}$; it follows that $\{f(w,\vec{y}\,/\,\vec{z})\colon w\in x\}$ is in $L^{\mathrm{tc}(\langle x\rangle*\vec{y}*\vec{z})}_{k\cdot(2+\mathrm{rk}(\langle x\rangle*\vec{y}))^n\cdot\mathrm{rk}(x)}$. 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 in $L^{\mathrm{tc}(\langle x\rangle*\vec{y}*\vec{z}*(\{f(w,\vec{y}\,/\,\vec{z})\colon w\in x\}))}_{(2+\mathrm{rk}(\langle x\rangle*\vec{y}))^n}$ and therefore by Lemma 4.2 also in $L^{\mathrm{tc}(\langle x\rangle*\vec{y}*\vec{z})}_{k\cdot(2+\mathrm{rk}(\langle x\rangle*\vec{y}))^n\cdot(\mathrm{rk}(x)+1)}$. This completes the induction step. That $f(x,\vec{y}\,/\,\vec{z})$ is uniformly definable in $x,\vec{y},\vec{z}$ follows immediately by considering

$$\bar{f}(x,\vec{y}\,/\,\vec{z})=\{(w,f(w,\vec{y}\,/\,\vec{z}))\colon w\in\mathrm{tc}(\{x\})\}$$

because we can express $\bar{f}(x,\vec{y}\,/\,\vec{z})=s$ as the formula

$$\mathrm{domain}(s)=\mathrm{tc}(\{x\})\quad\wedge\quad\forall w\in\mathrm{tc}(\{x\})\big(s(w)=h(w,\vec{y}\,/\,\vec{z},\{s(v)\colon v\in w\})\big),$$

using the uniform definition of $h(x,\vec{y}\,/\,\vec{z},u)$.

Now by choosing m large enough so that $k\cdot(2+\mathrm{rk}(\langle x\rangle*\vec{y}))^n\cdot(\mathrm{rk}(x)+1)$ is less than $(2+\mathrm{rk}(\langle x\rangle*\vec{y}))^m$ we have that $f(x,\vec{y}\,/\,\vec{z})$ is uniformly definable in $\mathrm{SR}_m(x,\vec{y}\,/\,\vec{z})$, as desired. ⊣

https://doi.org/10.1017/jsl.2015.26 Published online by Cambridge University Press

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

COROLLARY 4.6. *Suppose that $f(\vec{x}\,/\,)$ is safe recursive. Then for some finite n and some formula φ we have* :

1. *$f(\vec{x}\,/\,)$ belongs to $L^{\mathrm{tc}(\vec{x})}_{(2+\mathrm{rk}(\vec{x}))^n}$.*
2. *$f(\vec{x}\,/\,) = y$ if and only if $L^{\mathrm{tc}(\vec{x})}_{(2+\mathrm{rk}(\vec{x}))^n} \vDash \varphi(\vec{x}, y)$.*

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

COROLLARY 4.7. *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.

4.3. The lower bound result. Now, we aim for a converse of Theorem 4.5. 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 4.8. *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: $g^0(/\,x) = x$, $g^{\alpha+1}(/\,x) = g(/\,g^\alpha(/\,x))$, $g^\lambda(/\,x) = \bigcup_{\alpha<\lambda} g^\alpha(/\,x)$ 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 predicative set recursion, first define the function $k(x, y\,/\,)$ via a predicative set recursion as follows:

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

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

Recall from Section 2.2 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* if, 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 4.9.

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 $g(/\,x) = (x \cup (\bigcup x))$ $\mathrm{rk}(x)$ times. So the result follows from the previous lemma.
2. The function
$$g(/\,a) = a \cup \bigcup_{1 \leq i \leq 10} \mathrm{range}(G_i \restriction a \times a)$$

https://doi.org/10.1017/jsl.2015.26 Published online by Cambridge University Press

(see Lemma 4.3) belongs to SRud. It follows from the previous lemma that the function

$$g^*(T\,/\,) = \text{the closure of } T \cup \{T\} \text{ under } g$$

(restricted to transitive T) is safe recursive with parameter ω, as it 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 4.3 and 4.3, using the fact that ordinal multiplication is safe recursive. ⊣

We therefore get the following partial converse to Theorem 4.5.

THEOREM 4.10. *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 4.9 (4.9), 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 [7, 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 4.11. *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. Furthermore, when T is transitive of infinite rank, then ω belongs to the safe recursive closure of T. Therefore we have:

COROLLARY 4.12. *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:

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

COROLLARY 4.13. *For every α, $\mathrm{SR}_{1+\alpha} = L_{\omega^{\omega^\alpha}}$.*

To eliminate the parameter ω from Corollary 4.11 we redefine SR_n slightly, using a slower hierarchy for L^T. Define M^T_α inductively as follows:

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

https://doi.org/10.1017/jsl.2015.26 Published online by Cambridge University Press

This hierarchy is very close to Jensen's S-hierarchy, a refinement of his J-hierarchy (see [12, page 244]). We have the following (see [12, page 255]):

LEMMA 4.14. *For any transitive set* T:

1. T *is an element of* M_1^T.
2. *Each* M_α^T *is transitive and* $\alpha \leq \beta$ *implies* $M_\alpha^T \subseteq M_\beta^T$.
3. $\mathrm{Ord}(M_\lambda^T) = \mathrm{Ord}(T) + \lambda$ *for limit* λ.
4. $M_\alpha^T = L_\alpha^T$ *if* α *is* ω *or* $\omega \cdot \alpha = \alpha$. *In particular,* $M_{\mathrm{rk}(x)^\omega}^T = L_{\mathrm{rk}(x)^\omega}^T$ *if* x *has rank greater than* 1.

DEFINITION 4.15. For sequences $\vec{x}$, $\vec{y}$ and $0 < n \leq \omega$ we define $\mathrm{SR}_n^*(\vec{x} \,/\, \vec{y})$ to be $M_{(2+\mathrm{rk}(\vec{x}))^n}^{\mathrm{tc}(\vec{x}*\vec{y})}$.

Lemma 4.2 and Theorem 4.5 (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 versions of Corollary 4.9 (4.9), (4.9) in which L is replaced by M and SR is replaced by SR*: Whereas obtaining $L_{\alpha+1}^T$ from L_α^T requires a predicative set recursion of length ω, $M_{\alpha+1}^T$ is obtained from M_α^T by a single application of a function in SRud. In conclusion, we get the following characterization:

THEOREM 4.16. *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.

4.4. Safe recursive functions 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 $L_{\omega^n}^{\mathrm{tc}(x)}$ 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 relativized 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.

We obtain the following Theorem, where the equivalence between 4.17 and 4.17 is implicit in the analysis of the "Theory Machine", the universal infinite-time Turing machine considered in [9].

THEOREM 4.17. *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* [10]) *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* $\beta < \omega^\omega$.
3. $g(x) = G(x\,/\,)$ *for some* $G \in \mathrm{SRSF}$.

This shows 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

https://doi.org/10.1017/jsl.2015.26 Published online by Cambridge University Press

which are computable in polynomial time on an infinite-time Turing machine in the sense of [17].

§5. A machine model for safe recursive functions. 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 $\{(x, \beta, P_x^\beta)\colon \beta < \alpha\}$ by $P_x^{<\alpha}$. Now define:

$$P_x^\alpha = h(/\,P_{\in x}^{\leq \alpha} \cup P_x^{<\alpha}). \tag{5.1}$$

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

THEOREM 5.1. *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 predicative set 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 characterization of safe recursive functions given by Theorem 4.16, 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 (5.1) above. ⊣

§6. Acknowledgments. This research was partially done while the first author was a visiting fellow at the Isaac Newton Institute for the Mathematical Sciences in the programme "Semantics & Syntax". The second author was supported in part by NSF grants DMS-0700533, DMS-1101228, and CCR-1213151, and by the Simons Foundation grants 208717 and 306202. The third author was supported in part by the FWF (Austrian Science Fund) through FWF project number P 22430-N13. 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.

REFERENCES

[1] TOSHIYASU ARAI, *Predicatively computable functions on sets*. ***Archive for Mathematical Logic***, vol. 54 (2015), pp. 471–485.

[2] STEPHEN BELLANTONI and STEPHEN COOK, *A new recursion-theoretic characterization of the polytime functions*. ***Computational Complexity***, vol. 2 (1992), no. 2, pp. 97–110.

[3] LEONARD BERMAN, *The complexity of logical theories*, ***Theoretical Computer Science***, vol. 11 (1980), pp. 71–77.

https://doi.org/10.1017/jsl.2015.26 Published online by Cambridge University Press

[4] Lenore Blum, Mike Shub, and Steve Smale, *On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines*. ***Bulletin of the American Mathematical Society (N.S.)***, vol. 21 (1989), no. 1, pp. 1–46.

[5] Anna R. Bruss and Albert R. Meyer, *On the time-space classes and their relation to the theory of real addition*. ***Theoretical Computer Science***, vol. 11 (1980), pp. 59–69.

[6] Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer, *Alternation*, ***Journal of the Association for Computing Machinery***, vol. 28 (1981), pp. 114–133.

[7] Keith J. Devlin, ***Constructibility***, Perspectives in Mathematical Logic, Springer-Verlag, Berlin, 1984.

[8] Jeanne Ferrante and Charles W. Rackoff, *A decision procedure for the first order theory of real addition with order*. ***SIAM Journal on Computing***, vol. 4 (1975), no. 1, pp. 69–76.

[9] Sy-David Friedman and Philip D. Welch, *Two observations concerning infinite time Turing machines*, ***BIWOC 2007 Report*** (I. Dimitriou, editor), Hausdorff Centre for Mathematics, Bonn, January 2007, pp. 44–47.

[10] Joel David Hamkins and Andy Lewis, *Infinite time Turing machines*, this Journal, vol. 65 (2000), no. 2, pp. 567–604.

[11] Thomas Jech, ***Set Theory***, Springer Monographs in Mathematics, Springer-Verlag, Berlin, 2003.

[12] Ronald Björn Jensen, *The fine structure of the constructible hierarchy*. ***Annals of Mathematical Logic***, vol. 4 (1972), pp. 229–308.

[13] Ronald Björn Jensen and Carol Karp, *Primitive recursive set functions*, ***Axiomatic Set Thoory (Proceedings of Symposia in Pure Mathematics, vol. XIII, Part 1, University of California, Los Angeles, California, 1967)***, American Mathematical Society, Providence, R.I., 1971, pp. 143–176.

[14] Daniel Leivant, *Subrecursion and lambda representation over free algebras (preliminary summary)*, ***Feasible mathematics*** (S. Buss and P. Scott, editors), Birkhäuser, 1990, pp. 281–292.

[15] ———, *A foundational delineation of computational feasibility*, ***Proceedings of 6th Annual Symposium on Logic in Computer Science (LICS'91)***, IEEE Computer Society, 1991, pp. 2–11.

[16] Vladimir Yu. Sazonov, *On bounded set theory*, ***Logic and Scientific Methods (Florence, 1995)***, Synthese Library, vol. 259, Kluwer Academic Publishers, Dordrecht, 1997, pp. 85–103.

[17] Ralf Schindler, P $\neq$ NP *for infinite time Turing machines*. ***Monatshefte für Mathematik***, vol. 139 (2003), no. 4, pp. 335–340.

DEPARTMENT OF COMPUTER SCIENCE
COLLEGE OF SCIENCE
SWANSEA UNIVERSITY
SWANSEA SA2 8PP, UK
E-mail: a.beckmann@swansea.ac.uk

DEPARTMENT OF MATHEMATICS
UNIVERSITY OF CALIFORNIA
SAN DIEGO, LA JOLLA
CA 92093-0112, USA
E-mail: sbuss@ucsd.edu

KURT GÖDEL RESEARCH CENTER FOR MATHEMATICAL LOGIC
UNIVERSITY OF VIENNA
A-1090 VIENNA, AUSTRIA
E-mail: sdf@logic.univie.ac.at