Feasible set functions have small circuits

Arnold Beckmann
Department of Computer Science
Swansea University
a.beckmann@swansea.ac.uk

Sam Buss*
Department of Mathematics
University of California, San Diego
sbuss@ucsd.edu

Sy-David Friedman†
Kurt Gödel Research Center
University of Vienna
sdf@logic.univie.ac.at

Moritz Müller‡
Kurt Gödel Research Center
University of Vienna
moritz.mueller@univie.ac.at

Neil Thapen§
Institute of Mathematics
Czech Academy of Sciences
thapen@math.cas.cz

June 20, 2018

Abstract

The Cobham Recursive Set Functions (CRSF) provide an analogue of polynomial time computation which applies to arbitrary sets. We give three new equivalent characterizations of CRSF. The first is algebraic, using subset-bounded recursion and a form of Mostowski collapse. The second is our main result: the CRSF functions are shown to be precisely the functions computed by a class of uniform, infinitary, Boolean circuits. The third is in terms of a simple extension of the rudimentary functions by transitive closure and subset-bounded recursion.

*Supported in part by NSF grants DMS-1101228 and CCR-1213151, by the Simons Foundation, award 306202, and by the Skolkovo Institute for Science and Technology.

†Supported by the Austrian Science Fund (FWF) under project number P24654.

‡Supported by the Austrian Science Fund (FWF) under project number P28699.

§Partially supported by the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement 339691. The Institute of Mathematics of the Czech Academy of Sciences is supported by RVO:67985840.

1 Introduction

Computability over the natural numbers has over the years been successfully extended to robust notions of computability on ordinals, on objects of finite type and on sets in general (see for example Sacks' book [10]). Our goal is to develop an analogous, robust extension of computational complexity to arbitrary sets. This is the third of a series of papers exploring the model for polynomial-time computation on arbitrary sets given by the *Cobham recursive set functions* (CRSF) [4, 5]. A broader and promising future programme is to carry this out for other notions from complexity theory. This paper is largely self-contained. In particular, as it relies on a different definition of CRSF, it can be read independently of [4] and [5].

The class CRSF was introduced in [4] to capture the notion of feasible, polynomial time computation on arbitrary sets. In particular, it coincides with the usual polynomial time functions on finite binary strings, if strings in $\{0,1\}^k$ are identified with the corresponding set-theoretic functions in ${}^k2$.

The definition of CRSF in [4] is as a function algebra, based on a generalization of Cobham recursion on notation to arbitrary sets. A proof-theoretic characterization of CRSF, in terms of a version of Kripke-Platek set theory, is given in [5]. Furthermore, as shown in [4], the CRSF functions are closely connected to the Predicatively Computable Set Functions (PCSF) defined by Arai [2], which give a different characterization of polynomial time functions on sets. The PCSF functions are defined using safe recursion on notation, which was earlier used in [3] to define the larger class of Safe Recursive Set Functions (SRSF). A related notion of polynomial time on certain infinite sets was defined by Schindler [12]; see [3] for connections between this and SRSF. Sazonov [11] introduced a class of polynomial time functions on hereditarily finite sets, which we compare to CRSF in [4].

In [4] we take $\in$-recursion as the basic model for computation on sets. The innovation is that the power of $\in$-recursion is restricted by allowing new functions to be introduced only if their output is no more complex than the output of a function already known to be in CRSF, in the style of Cobham's definition of polynomial time [6]. Here a set a is no more complex than a set b if a is embeddable in b in a certain sense (which we describe later). To allow a limited, "polynomial" increase in complexity [4] adapts the smash function # of bounded arithmetic into an operation on sets, namely a kind of cartesian product on Mostowski graphs, and includes this as one of the initial functions.

We introduce here three alternative characterizations of CRSF. These all take $\in$-recursion as fundamental, but they restrict its strength in different

ways, and one of them does not use the smash function. That they all give rise to the same class of functions gives more evidence that CRSF is natural.

The first characterization is similar to [4]. The class $\mathrm{CRSF}_{\subseteq}$ is formed by taking some basic initial functions, including the smash function, and closing under composition and *subset-bounded recursion*: if g and h are in the class, then so is the function f defined by the recursion

$$f(\vec{a}, b) = g(\vec{a}, b, \{f(\vec{a}, c) : c \in b\}) \cap h(\vec{a}, b).$$

We call this "subset-bounded" because it allows defining a function f by recursion only if we have in hand a function h such that $f(\vec{a}, b) \subseteq h(\vec{a}, b)$. The main difference between this and the definition in [4] is the use of this simpler kind of recursion instead of the rather complicated "embedding-bounded" recursion of [4], of which it is a special case. The disadvantage is that $\mathrm{CRSF}_{\subseteq}$ functions are limited in what they can output, because any output value must look more or less like a subset of (some basic function of) the input.

To deal with this we use a system for coding sets as subsets. A standard way to represent a finite graph in computer science is as a subset of $n \times n$ (coded as a binary string of length n^2) giving the edge relation, where n is a size parameter. Similarly we can take a set, represent a copy of its Mostowski graph as a subset E of $a \times a$, where a is some suitable set, and then recover the original set from E using Mostowski collapse. It turns out that we still get a robust system of coding if we restrict the kinds of subset E that can appear (by only allowing edges consistent with the ordering induced by $\in$), and that then we do not need the full strength of Mostowski collapse but can make do with a limited, "feasible" version of it.

We define $\mathrm{CRSF}^{+}_{\subseteq}$ by adding this limited Mostowski collapse function to $\mathrm{CRSF}_{\subseteq}$. We show that $\mathrm{CRSF}^{+}_{\subseteq}$ and the original CRSF are the same, and in particular that for every CRSF function $f(\vec{x})$ there is a $\mathrm{CRSF}_{\subseteq}$ function which computes a code for $f(\vec{x})$. It follows that we can compute $f(\vec{x})$ in $\mathrm{CRSF}^{+}_{\subseteq}$ with only a single use of Mostowski collapse, and also that $\mathrm{CRSF}_{\subseteq}$ and CRSF contain the same relations, that is, the same 0/1 valued functions.

Our second characterization takes a very different approach, by describing a Boolean circuit model of computation on sets. We define (possibly infinite) circuits, which act on Boolean (0/1) values and have the usual conjunction, disjunction and negation gates. To allow these to input and output sets, we use the method of coding outlined above. For example, if we want to take as input sets which can be coded as a subset E of $a \times a$, we include an input node for each member u of $a \times a$, and assign it the value 1 if $u \in E$

and 0 otherwise – and as is usual in circuit complexity, we will need a different circuit for each size parameter a. We show that CRSF can be precisely characterized as the functions which can be computed by strongly uniform families of small Boolean circuits, where "small" is defined in terms of the smash function. This is our main result, and shows that a basic property of polynomial time functions carries across smoothly to arbitrary sets.

There are several advantages to the Boolean circuit characterization. First, it is quite different from the earlier characterizations, thus providing more evidence of the robustness of CRSF. Second, it makes clear that CRSF is, in part, a model of parallel computation. This fact is obscured in the earlier development in [4], as that work focused on the equivalence with polynomial time computation. Third, it allows tools from the usual theory of Boolean circuit complexity to be applied to CRSF. As an example, using the Hastad switching lemma about the inexpressibility of parity in AC^0 [8], we can show a version of P$\neq$NP for CSRF. We hasten to mention that this does not say anything about whether the usual classes of P and NP are distinct.

Our third characterization is again as a function algebra, this time defined by extending the rudimentary functions in an elementary way. We take the class RS to be the rudimentary functions plus the transitive closure function, all closed under subset-bounded recursion. We do not add the smash function, and it is easy to see that smash is not in RS, which hence is different from CRSF. However we adapt our system of coding to RS and show that CRSF functions can be defined in RS via their codes, and thus that the classes are essentially the same except for issues of decoding. The key is to show that $\in$-recursion in the presence of smash can be simulated by lexicographic $\in$-recursion without smash.

There are several potential routes for future work. This paper, together with related work [3, 2, 4, 12], shows that we have established a robust understanding of polynomial time in set theory. Obvious next steps are to find set theoretic analogues for other time, space or circuit complexity classes which are well-known and well-studied in the context of finite binary strings.

The outline is as follows. Section 2 discusses preliminary topics, and introduces our version of the Mostowski collapse and smash functions. Section 3 introduces subset-bounded recursion, defines our version of CRSF, and proves some basic properties of these definitions. Section 4 introduces a simple model of infinite-time Turing machine computation and shows that on finite and infinite binary strings, CRSF functions are the same as those com-

puted by corresponding notions of polynomial time Turing machines. Section 5 introduces some technical material, in particular bisimilarity, which we will use to detect when two encodings of Mostowski graphs represent the same set. Section 6 defines infinite Boolean circuits, and also the notion of $\Delta_0^{\#}$-uniform families of Boolean circuits. Section 7 proves a series of strong technical results about the power of $\Delta_0^{\#}$-uniform circuits. Section 8 completes the proof that the CRSF functions are precisely the functions computable with $\Delta_0^{\#}$-uniform circuits. It also gives a P$\neq$NP style result for CRSF functions acting on hereditarily finite (HF) sets. Section 9 proves the equivalence of our version of CRSF and CRSF as defined in [4]. Section 10 shows that adding transitive closure and subset-bounded recursion to Jensen's rudimentary functions gives another characterization of CRSF.

2 Preliminaries

2.1 Notational conventions

The partial ordering induced on sets by $\in$ will play a fundamental role for us, analogous to the ordering on natural numbers. We therefore use the notation $<$ for the transitive closure of the $\in$ relation, and the notation $\leq$ for the reflexive transitive closure. Writing $\mathrm{tc}(b)$ for the transitive closure of b, this means that $a < b$ and $a \leq b$ are equivalent to $a \in \mathrm{tc}(b)$ and $a \in \mathrm{tc}(\{b\})$, respectively. To further strengthen the analogy to the interval notation and because it will be convenient for the generalized notion of "binary string" defined below, we write $[a]$ for $\mathrm{tc}(\{a\})$ (this was denoted $\mathrm{tc}^+(a)$ in [4]). This notation is meant to suggest the "interval" $[0, a] = \{x : \emptyset \leq x \leq a\}$.

We often code one set inside another set in a way that generalizes the usual notion of binary strings. We will stick as much as possible to the following convention. We treat some sets as "raw materials", inside whose Mostowski graphs we will construct other objects. We write these "raw material" sets using small letters $a, b, \ldots$; they are analogous to unary strings in complexity theory. We use capital letters $E, F, \ldots$ or $U, V, \ldots$ for objects we construct as subsets of the Mostowski graphs of sets of the first kind; these are analogous to binary strings. For ordinary binary strings, the analogy is precise: an ordinary unary string of length n is identified with the von Neumann integer $a = n$, and an ordinary binary string of length n is then the subset U of a having as members the positions where a bit 1 appears in the string.

In a directed graph, if there is an edge from a node u to a node v we say that u is a *predecessor* of v. If there is a path (possibly of length 0)

from u to v we say that u is an *ancestor* of v. We will usually be dealing with acyclic (in fact well-founded) directed graphs, and when we describe directed graphs we think of the edges as pointing upwards.

We define the ordered pair as $\langle a, b\rangle = \{\{a\}, \{a, b\}\}$ and extend this in the usual way to ordered k-tuples, with $\langle a, b, c\rangle = \langle a, \langle b, c\rangle\rangle$ etc. and $\langle \vec{a}\rangle = a$ if $\vec{a}$ is a 1-tuple. We will write $[a]^k$ for the k-th cartesian power of $[a]$, since we will need to refer to this often. But we do not use this notation for sets not written in the form $[a]$ to avoid confusion with ordinal exponentiation.

We identify the natural numbers with the finite von Neumann ordinals $\emptyset, \{\emptyset\}, \ldots$ For the sake of consistency we will always write 0 instead of $\emptyset$.

2.2 Embedded Mostowski collapse

Recall that a directed graph with nodes U and edges E is *well-founded* if, for every non-empty subset S of U, there is a node y in S with no predecessors in S. It is *extensional* if no two distinct nodes have the same set of predecessors. We say that it is *accessible pointed* with sink a if there is a path from every node to a.[1]

The *Mostowski graph* of a set a is the directed graph $\mathcal{G}(a) := \langle [a], E\rangle$ with nodes $[a]$ and with edges $E = \{\langle x, y\rangle \in [a]^2 : x \in y\}$. Clearly $\mathcal{G}(a)$ is well-founded, extensional and accessible pointed, with sink a. By well-foundedness and extensionality it also has exactly one source node, the empty set. On the other hand if G is any well-founded, extensional, accessible pointed graph then there is a unique set a, the *Mostowski collapse* of G, such that G is isomorphic to $\mathcal{G}(a)$. We denote this set a by $\mathcal{M}(G)$.

Definition 2.1 *A* diagram *is a pair $\langle a, E\rangle$ of sets such that $\langle x, y\rangle \in E$ only if $x < y$.*

The diagram $\langle a, E\rangle$ represents the graph with nodes $[a]$ and edges $E \cap [a]^2$. We think of $\langle a, E\rangle$ as a graph "embedded" in the Mostowski graph $\mathcal{G}(a)$ of a, with edges that are forced to respect the ordering on $\mathcal{G}(a)$. An example of a diagram is the pair $\langle a, \in\restriction[a]^2\rangle$, representing $\mathcal{G}(a)$ itself.

A diagram is automatically well-founded, by the condition on E. In general it is not accessible pointed or extensional; for example any nodes in $[a]$ outside the range of E will have the same, empty set of predecessors. However, if we restrict the graph to the set of nodes that are E-ancestors of a, then it is accessible pointed, with sink a. Furthermore, we can make

[1] Aczel [1] defines an *accessible pointed graph* as one with a distinguished node from which every other node is reachable. Note that our paths run in the opposite direction.

it extensional by recursively collapsing together nodes with the same set of predecessors. We can then take the Mostowski collapse of the resulting graph. This procedure, which we call *embedded Mostowski collapse*, has a simple recursive definition:

Definition 2.2 *The embedded Mostowski collapse function $M(a, E)$ is defined by*

$$M(a, E) = \{M(b, E) : b < a \wedge \langle b, a\rangle \in E\}.$$

Notice that this is definable by $\in$-recursion. We will almost always use Mostowski collapse in this form, so will often omit the word "embedded".

We use $\mathrm{rank}(x)$ to denote the von Neumann rank of the set x and $|x|$ to denote its cardinality.

Lemma 2.3 $|[M(a, E)]| \leq |[a]|$ *and* $\mathrm{rank}(M(a, E)) \leq \mathrm{rank}(a)$.

Definition 2.4 *We say* b is embeddable in a *if* $b = M(a, E)$ *for some* E.

We interpret the embeddability of b in a as meaning that b is no more complex than a, in the sense that, for example, recursion over b is no more powerful than recursion over a. Lemma 9.4 shows that our notion of embeddability is the same as the one in [4].

2.3 The smash function

We repeat some definitions from [4].

Definition 2.5 *The* set composition function $a \odot b$ *is defined as follows. Given sets a and b, construct a graph H by drawing the graph $\mathcal{G}(a)$ above the graph $\mathcal{G}(b)$ and then identifying the source of $\mathcal{G}(a)$ with the sink of $\mathcal{G}(b)$. Then $a \odot b = \mathcal{M}(H)$.*

An equivalent recursive definition for set composition is:

$$a \odot b = \begin{cases} b & \text{if } a = 0 \\ \{x \odot b : x \in a\} & \text{if } a \neq 0. \end{cases}$$

Definition 2.6 *The* set smash function $a\#b$ *is defined as follows. Given sets a and b, construct a graph H by drawing a disjoint copy G_x of $\mathcal{G}(b)$ for every point $x \in \mathcal{G}(a)$, and then adding an edge from the sink of G_x to the source of G_y for every edge $\langle x, y\rangle$ of $\mathcal{G}(a)$. Then $a\#b = \mathcal{M}(H)$.*

An equivalent recursive definition for set smash is:

$$a\#b = b \odot \{x\#b : x \in a\}.$$

The smash function is a kind of cartesian product on Mostowski graphs. We introduce a corresponding "pairing function" $\sigma_{a,b}(x, y)$, which we think of as taking nodes $x \in \mathcal{G}(a)$ and $y \in \mathcal{G}(b)$ and outputting the node corresponding to y in the x-th copy G_x of $\mathcal{G}(b)$ in H. Note that although we write a as a subscript in $\sigma_{a,b}$ we do not actually use a in the definition.

Definition 2.7 *We define* $\sigma_{a,b}(x, y) = y \odot \{z\#b : z \in x\}$.

Lemma 2.8 *The function* $\langle x, y\rangle \mapsto \sigma_{a,b}(x, y)$ *is an order isomorphism between* $[a] \times [b]$*, ordered lexicographically, and* $[a\#b]$.

Definition 2.9 *We let* $\pi_{1,a,b} : [a\#b] \to [a]$ *and* $\pi_{2,a,b} : [a\#b] \to [b]$ *be projection functions inverting* $\sigma_{a,b}$*, so that* $\sigma_{a,b}(\pi_{1,a,b}(z), \pi_{2,a,b}(z)) = z$ *for* $z \in [a\#b]$.

Lemma 2.10 *For sets* a *and* b*,*

1. $\mathrm{rank}(a \odot b) = \mathrm{rank}(b) + \mathrm{rank}(a)$
2. $|\mathrm{tc}(a \odot b)| = |\mathrm{tc}(a)| + |\mathrm{tc}(b)|$
3. $\mathrm{rank}(a\#b) + 1 = (\mathrm{rank}(b) + 1)(\mathrm{rank}(a) + 1)$
4. $|\mathrm{tc}(a\#b)|+1 = (|\mathrm{tc}(b)|+1)(|\mathrm{tc}(a)|+1)$*, equivalently,* $|[a\#b]| = |[a]|\cdot|[b]|$.

Definition 2.11 *A* smash-term *is a term built from variables, the constant* 0 *and the functions pairing, cartesian product, transitive closure,* $\odot$ *and* $\#$.

Smash-terms will play the role usually played by polynomials in computational complexity, providing bounds for various complexity measures. Notice that the rank, and respectively the size of the transitive closure, of a smash-term is at most polynomially larger than those of its arguments. (The corresponding definition of #-term in [4] is stricter, only allowing variables, the constant 1, $\odot$ and $\#$.)

3 Subset-bounded recursion and CRSF

This section is modelled on the similar development of CRSF in [4].

Definition 3.1 *Let $g(\vec{a}, b, x)$ and $h(\vec{a}, b)$ be functions from sets to sets. The function $f(\vec{a}, b)$ is obtained from $g(\vec{a}, b, x)$ by* subset-bounded recursion *with bound $h(\vec{a}, b)$ if*

$$f(\vec{a}, b) \;=\; g(\vec{a}, b, \{f(\vec{a}, c) : c \in b\}) \cap h(\vec{a}, b).$$

Definition 3.2 *We take as initial functions*

1. *the constant* 0
2. *projection:* $a_1, \ldots, a_n \mapsto a_j$ *for* $1 \leq j \leq n$
3. *pairing:* $a, b \mapsto \{a, b\}$
4. *union:* $a \mapsto \bigcup a$
5. *conditional:* $\mathrm{cond}_{\in}(a, b, c, d) = \begin{cases} a & \text{if } c \in d \\ b & \text{otherwise} \end{cases}$
6. *transitive closure:* $a \mapsto \mathrm{tc}(a)$
7. *cartesian product:* $a, b \mapsto a \times b$
8. *set composition:* $a, b \mapsto a \odot b$
9. *set smash:* $a, b \mapsto a \# b$
10. *embedded Mostowski collapse:* $a, E \mapsto M(a, E)$.

The above initial functions are roughly the same as the primitive symbols in the definition of Cobham recursive set functions given in [5], but with the addition of the embedded Mostowski collapse function.

Definition 3.3 $\mathrm{CRSF}_{\subseteq}$ *is defined as the closure of initial functions 1 to 9 under composition and subset-bounded recursion.* $\mathrm{CRSF}^{+}_{\subseteq}$ *is defined as the closure of all of the initial functions above, including embedded Mostowski collapse, under composition and subset-bounded recursion.*

Definition 3.4 *A* $\mathrm{CRSF}_{\subseteq}$ relation *is a relation $\varphi(\vec{a})$ given by an expression of the form $g(\vec{a}) \neq 0$ where g is a* $\mathrm{CRSF}_{\subseteq}$ *function. The* $\mathrm{CRSF}^{+}_{\subseteq}$ *relations are defined similarly.*

Lemma 3.5 *We derive some basic properties of* $\mathrm{CRSF}_{\subseteq}$*. The same properties also hold with* $\mathrm{CRSF}^{+}_{\subseteq}$ *in place of* $\mathrm{CRSF}_{\subseteq}$*.*

1. $\mathrm{CRSF}_{\subseteq}$ *contains the functions* $\{a\}$, $a \cup b$ *and*

$$\mathrm{cond}_{=}(a,b,c,d) \;=\; \begin{cases} a & \text{if } c = d \\ b & \text{otherwise.} \end{cases}$$

2. *The* $\mathrm{CRSF}_{\subseteq}$ *relations are closed under Boolean operations.*
3. $\mathrm{CRSF}_{\subseteq}$ *is closed under* separation. *That is, if* $\varphi(\vec{a}, c)$ *is a* $\mathrm{CRSF}_{\subseteq}$ *relation then the following function is in* $\mathrm{CRSF}_{\subseteq}$*:*

$$f(\vec{a}, b) = \{c \in b : \varphi(\vec{a}, c)\}.$$

4. *The* $\mathrm{CRSF}_{\subseteq}$ *relations are closed under* Δ_0 *quantification, in which quantifiers range over members of a given set.*
5. $\mathrm{CRSF}_{\subseteq}$ *contains the functions* $\bigcap a$, $a \setminus b$ *and* $a \cap b$. *By convention* $\bigcap 0 = 0$.
6. $\mathrm{CRSF}_{\subseteq}$ *contains the usual pairing and projection functions for* k*-tuples, for* $k \in \mathbb{N}$.
7. $\mathrm{CRSF}_{\subseteq}$ *contains* $\sigma_{a,b}$ *and the projection functions* $\pi_{1,a,b}$ *and* $\pi_{2,a,b}$.

Proof

1. We use $\{a\} = \{a, a\}$ and $a \cup b = \bigcup\{a, b\}$. We define $\mathrm{cond}_{=}$ as $\mathrm{cond}_{\in}(a, b, c, \{d\})$.
2. We define $\neg(g(\vec{a}) \neq 0)$ and $(f(\vec{a}) \neq 0) \vee (g(\vec{b}) \neq 0)$ respectively by $\mathrm{cond}_{=}(1, 0, g(\vec{a}), 0) \neq 0$ and $f(\vec{a}) \cup g(\vec{b}) \neq 0$.
3. Using $\mathrm{cond}_{\in}$, $\mathrm{cond}_{=}$ and Boolean operations we can define functions by cases. Define $k(\vec{a}, b, c)$ by recursion on c as

$$k(\vec{a}, b, c) \;=\; \begin{cases} \{c\} & \text{if } c \in b \text{ and } \varphi(\vec{a}, c) \\ \bigcup\{k(\vec{a}, b, d) : d \in c\} & \text{if } c = b \\ 0 & \text{otherwise.} \end{cases}$$

   Then $k(\vec{a}, b, c) \subseteq b$ always, so k is definable by subset-bounded recursion with bound b. We put $f(\vec{a}, b) = k(\vec{a}, b, b)$.
4. We can define the relation $\exists c{\in}b\, \varphi(\vec{a}, c)$ by

$$\{c \in b : \varphi(\vec{a}, c)\} \neq 0$$

   where the set on the left is given by separation.

5. We take $\bigcap a = \{x \in \bigcup a : \forall b{\in}a\,(x \in b)\}$, using separation. The other two are trivial.
6. Trivial.
7. We have $\sigma_{a,b}(x,y) = y \odot \{z\#b : z \in x\}$, where the set $\{z\#b : z \in x\}$ can be obtained by separation, since it is a subset of $\mathrm{tc}([x\#b])$. For $z \in [a\#b]$ we can define, for example, the projection function $\pi_{1,a,b}$ by

$$\pi_{1,a,b}(z) = \bigcup\{x \in [a] : \exists y{\in}[b]\,\sigma_{a,b}(x,y) = z\}$$

since exactly one x satisfies the condition on the right. □

Lemma 3.6 *The rank function is in* $\mathrm{CRSF}^+_\subseteq$.

Proof Given a set a, define $\rho(a)$ by first letting H be the transitive closure of $\mathcal{G}(a)$ as a graph, that is, we start with $\mathcal{G}(a)$ and add an edge $\langle x,y\rangle$ whenever there is a path from x to y, and then letting $\rho(a)$ be the Mostowski collapse of H. Then $\rho(a)$ is an ordinal, since it is a transitive set of transitive sets. Furthermore, we can show by induction on a that $\mathrm{rank}(\rho(a)) = \mathrm{rank}(a)$. We have $\rho(a) = \{\rho(x) : x \in \mathrm{tc}(a)\}$ so

$$\begin{aligned}\mathrm{rank}(\rho(a)) &= \sup\{\mathrm{rank}(y)+1 : y \in \rho(a)\}\\ &= \sup\{\mathrm{rank}(\rho(x))+1 : x \in \mathrm{tc}(a)\}\\ &= \sup\{\mathrm{rank}(x)+1 : x \in \mathrm{tc}(a)\}\\ &= \mathrm{rank}(a)\end{aligned}$$

Hence $\rho(a) = \mathrm{rank}(a)$. This construction can be done in $\mathrm{CRSF}^+_\subseteq$ by defining $\rho(a) = M(a,H)$ where $H = \{\langle x,y\rangle \in [a]^2 : x < y\}$. □

Note that the graph H above is in general not extensional — for example, consider what happens to the nodes $\{1\}$ and 2 when $a = \{\{1\},2\}$. This situation is similar to the appearance of a multi-valued embedding in the construction of the rank function in [4].

Lemma 3.7 *Ordinal addition and multiplication are in* $\mathrm{CRSF}^+_\subseteq$.

Proof Recall from Lemma 2.10 that $\mathrm{rank}(a \odot b) = \mathrm{rank}(b) + \mathrm{rank}(a)$ and $\mathrm{rank}(a\#b) + 1 = (\mathrm{rank}(b)+1)(\mathrm{rank}(a)+1)$. Hence the function $a + b = \mathrm{rank}(b{\odot}a)$ gives us ordinal addition, and we can define ordinal multiplication by subset-bounded recursion as

$$a \cdot b = \bigcup\{a \cdot c + a : c \in b\} \cap \mathrm{rank}(b\#a)$$

since $a \cdot b \subseteq \mathrm{rank}(b\#a)$ for ordinals. □

Lemma 3.8 *For any* $\mathrm{CRSF}^+_\subseteq$ *function* f*, there is a polynomial* p *such that* $\mathrm{rank}(f(\vec{a})) \leq p(\mathrm{rank}(\vec{a}))$ *and* $|[f(\vec{a})]| \leq p(|[\vec{a}]|)$.

Proof This is a straightforward induction on the complexity of f. □

Corollary 3.9 *Ordinal exponentiation is not in* $\mathrm{CRSF}^+_\subseteq$.

Corollary 3.10 *There is no* $\mathrm{CRSF}^+_\subseteq$ *function which, on all hereditarily finite sets* x*, ouputs the ordinal* $|x|$.

Proof Let f be such a function. Take $n \in \mathbb{N}$ and let a be its power set $\mathbb{P}(n)$, so $f(a)$ is the ordinal 2^n. Then $\mathrm{rank}(a) = n + 1$ while $\mathrm{rank}(f(a)) = 2^n$. □

We conclude this section with a digression about the choice of initial functions and whether they are all necessary. It turns out that Mostowski collapse plays a more limited role in $\mathrm{CRSF}^+_\subseteq$ than might be expected. We will show in Section 8 that for any function $f(\vec{a})$ in $\mathrm{CRSF}^+_\subseteq$, there is a smash-term $t(\vec{a})$ and a function $g(\vec{a})$ in $\mathrm{CRSF}_\subseteq$ such that $f(\vec{a}) = M(t(\vec{a}), g(\vec{a}))$. In other words, if we can compute a set in $\mathrm{CRSF}^+_\subseteq$, then we can already compute a diagram of it in $\mathrm{CRSF}_\subseteq$. In particular, Mostowski collapse is not needed if we are only interested in computing 0/1-valued functions.

We expect that we could do without cartesian product, since if we want to quantify over pairs in $a \times b$ we could instead quantify over elements of $[a\#b]$, using the function $\sigma_{a,b}(x, y)$ where we now use the ordered pair $\langle x, y\rangle$. This change would require some formal changes to our definitions of diagrams and Mostowski collapse, to make use of this new system.

In the full system $\mathrm{CRSF}^+_\subseteq$ we do not need to include set composition as an initial function, since it is easy to construct the Mostowski graph of $a \odot b$ as a diagram embedded inside $a\#b$ and then recover $a \odot b$ using Mostowski collapse.

The smash function would seem to play a central role in introducing polynomial growth rate functions. Nonetheless, there is a natural way to extend embedded Mostowski collapse which removes the need for the smash function. For $k \in \mathbb{N}$, define *k-embedded Mostowski collapse* as

$$M_k(\langle\vec{a}\rangle, E) \;=\; \{M_k(\langle\vec{b}\rangle, E) : \langle\vec{b}\rangle <_k \langle\vec{a}\rangle \wedge \langle\langle\vec{b}\rangle, \langle\vec{a}\rangle\rangle \in E\}$$

where $\langle\vec{a}\rangle, \langle\vec{b}\rangle$ are k-tuples and $<_k$ is the lexicographic ordering given by $<$. We can use the order-isomorphism between $[a]^k$ under $<_k$ and $[a\# \cdots \#a]$ (with k occurrences of a) under $<$ to define M_k in $\mathrm{CRSF}^+_\subseteq$. In the other direction, it is straightforward to define the smash function using M_2, so the class $\mathrm{CRSF}^+_\subseteq$ is unchanged if we remove smash and set composition and replace M with M_2. For more on $<_k$ and M_k, see Section 10.

4 Turing machines

We consider a simple model of infinite-time Turing machines [7]. The usual finite Turing machines are special cases of the definition below, obtained by considering only finite ordinals and skipping any text containing the word "limit".

We simulate machines operating on strings of symbols from a finite alphabet of numerals $0, \ldots, k-1$. These strings may be finite, but more generally will have ordinal length. We need to specify how to code such strings as sets, so that we can manipulate them with $\mathrm{CRSF}^+_\subseteq$ functions. We do this in a straightforward way by letting a string of length $\lambda \leq \omega$ be formally a function $\lambda \to k$ and dealing with this directly as a set-theoretic function, that is, as a set of ordered pairs.[2]

Consider a Turing machine A with a single, one-way tape, m states, and k symbols. We will simulate the machine running on a tape of length $\lambda \leq \omega$ for time τ, where λ and τ are ordinals (that is, λ will be either finite or ω). A configuration of the machine is a triple $\langle W, i, s\rangle$ where the string W is the contents of the tape, i is the position of the head and s is the state.

At each step $\sigma + 1$, machine A reads the symbol that was under the head in step σ, then writes a symbol, changes state and moves the head, all according to the transition function (as usual).

At limit steps σ, we set the symbol in each cell j in W to be the highest symbol that occurs cofinally often in cell j as we range over the earlier configurations. We change the state s to a distinguished limit state and move the head to $i = 0$.

Let $\mathrm{Config}_A(\lambda, I, \tau)$ be the function that takes as inputs ordinals $\lambda \leq \omega$ and τ, and a string I of ordinal length $\leq \lambda$, and outputs the configuration of the machine A with tape length λ after running for τ steps on input I.

Lemma 4.1 Config_A *is in* $\mathrm{CRSF}_\subseteq$.

Proof Config_A can be defined by a straightforward recursion. The only technical issue is to make sure that the intermediate values in the recursion

[2]There are other ways of handling this. In particular, over the binary alphabet we could code strings of length λ simply as subsets of λ. We do not use this encoding here, partly to avoid issues of how to mark the end of a string, but we will use it beginning in Section 5.1 where we will be dealing with strings of fixed size. An alternative where we explicitly record the length λ would be to code binary strings $a_0 a_1 \ldots$ as pairs $\langle \lambda, \{i < \lambda : a_i = 1\}\rangle$. This would be essentially equivalent to the coding used in the current section, since there are $\mathrm{CRSF}^+_\subseteq$ functions translating in both directions between them.

are all subsets of a set we can construct. We assume that λ and τ are ordinals and will write $<$ rather than $\in$ for membership in an ordinal.

Let $F(\lambda, I, \tau)$ be the function that, on well-formed inputs, outputs

$$\{\tau\} \times W \times \{i\} \times \{s\} \;=\; \{\langle \tau, a, i, s\rangle : a \in W\}$$

where $\langle W, i, s\rangle$ is the configuration of M at step τ. Then

$$F(\lambda, I, \tau) \subseteq \{\tau\} \times (\lambda \times k) \times \lambda \times m$$

so we can potentially define F directly by subset-bounded recursion. To do so, we must show how to compute $F(\lambda, I, \tau)$ from $S = \{F(\lambda, I, \sigma) : \sigma < \tau\}$. Observe that for $\sigma < \tau$ we can recover $F(\lambda, I, \sigma)$ from S as the subset of $\bigcup S$ consisting of elements with first component σ. In particular we can recover the tape contents W_σ at step σ by separation as

$$W_\sigma = \{a \in \lambda \times k \,:\, \exists i{<}\lambda\, \exists s{<}m\, \langle \sigma, a, i, s\rangle \in \bigcup S\}$$

and can similarly recover the head position i_σ and the state s_σ.

If $\tau = 0$ then we let $i = 0$, let s be the starting state, and let W be the input string I, padded out as necessary to length λ with pairs $\langle j, 0\rangle$ (assuming without loss of generality that the symbol 0 stands for "blank"). If $\sigma := \bigcup \tau \in \tau$ then τ is the successor ordinal $\sigma + 1$. Using W_σ, i_σ, s_σ and the transition function of M, we compute W using separation and change i and s appropriately.

Otherwise τ is a limit. We let s be the limit state and let $i = 0$. The set of symbols occurring cofinally often in cell j is

$$X_j = \{x < k : \forall \sigma{<}\tau\, \exists \sigma'{<}\tau\, (\sigma < \sigma' \wedge \langle j, x\rangle \in W_{\sigma'})\}.$$

Hence $\bigcup X_j$ is the maximum symbol that occurs cofinally often in cell j, and we can let $W = \{a \in \lambda \times k : a = \langle j, \bigcup X_j\rangle\}$.

This shows that F is in $\mathrm{CRSF}_{\subseteq}$. The lemma follows. □

As usual, $\{0,1\}^*$ denotes the set of finite binary strings.

Theorem 4.2 *Every polynomial time function from $\{0,1\}^*$ to $\{0,1\}^*$ is in $\mathrm{CRSF}^+_{\subseteq}$.*

Proof Suppose f is computed by a Turing machine A which halts in time n^c on inputs of length $n > 1$. Given a finite string w, in $\mathrm{CRSF}^+_{\subseteq}$ we can compute the length n of w as $\{i \in \bigcup\bigcup w : \exists x{<}k\, \langle i, x\rangle \in w\}$. We then compute n^c (using Lemma 3.7) and $\mathrm{Config}_A(n^c, w, n^c)$, and the output of f can easily be obtained from this. □

The only use of Mostowski collapse in the proof of Theorem 4.2 is to obtain n^c from n. If we instead let $m = n\# \cdots \# n$ (where n appears c times) then m is order-isomorphic to $(n+1)^c - 1$, and we can prove a version of Lemma 4.1 that avoids using Mostowski collapse, by simulating Turing machine computations of length n^c using $\in$-recursion on m rather than on n^c. All that prevents us from carrying out the whole proof of the theorem in this way is that, under our coding of binary strings as sets, we must be able to convert the output into a sequence indexed by an ordinal. This is not a problem for very simple outputs, so we have:

Theorem 4.3 *Every polynomial time relation on* $\{0,1\}^*$ *is in* $\mathrm{CRSF}_{\subseteq}$.

As a converse of Theorem 4.2 we have the following.

Theorem 4.4 *Every* $\mathrm{CRSF}^+_{\subseteq}$ *function from* $\{0,1\}^*$ *to* $\{0,1\}^*$ *is in polynomial time.*

Proof We use the same argument as [4]. The theorem follows from the more general observation that for any $\mathrm{CRSF}^+_{\subseteq}$ function $f(x_1, \ldots, x_r)$ there is a polynomial time function which, for any hereditarily finite sets $a_1, \ldots, a_r$, given graphs $\mathcal{G}(a_1), \ldots, \mathcal{G}(a_r)$ outputs $\mathcal{G}(f(a_1, \ldots, a_r))$ (up to graph isomorphism) using the standard encoding of directed graphs as strings. This is proved by induction on the complexity of f. The bound on recursion guarantees that the sizes of the graphs involved do not grow too fast. □

We now move to infinite-time machines. Following [12], a function from $\{0,1\}^\omega$ to $\{0,1\}^\omega$ is polynomial time if it is computed by an infinite-time Turing machine with three tapes (an input tape, an output tape and a working tape) which always halts after at most ω^d steps, for some fixed exponent $d \in \mathbb{N}$.

Theorem 4.5 *Every polynomial time function from* $\{0,1\}^\omega$ *to* $\{0,1\}^\omega$ *is in* $\mathrm{CRSF}_{\subseteq}$.

Proof This is proved in the same way as Theorem 4.3, with minor changes to accommodate simulating three tapes rather than one. Since the output sequence is a subset of $\omega \times 2$, we can use separation to construct it from the output of Config_M and do not need Mostowski collapse. □

Theorem 4.6 *Every* $\mathrm{CRSF}^+_{\subseteq}$ *function from* $\{0,1\}^\omega$ *to* $\{0,1\}^\omega$ *is polynomial time.*

Proof It is shown in [3] that every such function from $\{0,1\}^\omega$ to $\{0,1\}^\omega$ in the class SRSF is polynomial time. By results in [4] and in Section 9 below, every function in $\mathrm{CRSF}^+_\subseteq$ is in SRSF. □

5 Bisimilarity and coding

This section discusses the generalized notion of binary strings, then bisimulation, and then coding collections of sets.

5.1 a-strings

Definition 5.1 *Let a be any set. An* a-string *is a subset of a.*

We will sometimes informally identify an a-string with its characteristic function $a \to \{0,1\}$. In this sense, for finite ordinals k, the usual binary strings in $\{0,1\}^k$ of complexity theory correspond to the k-strings. For example, the binary string $01101 \in \{0,1\}^5$ corresponds to the 5-string $\{0,2,3\}$ (writing the most significant bit on the left).

For us, the most important use of a-strings is to encode sets via a Mostowski collapse:

Definition 5.2 *We say that a diagram $\langle a, E\rangle$* codes *the set $M(a,E)$. If a is fixed and E is an $[a]^2$-string, we say E* codes *the set $M(a,E)$.*

This way of coding sets, as a size parameter a together with an $[a]^2$-string E, is designed to work well with subset-bounded recursion.

5.2 Bisimilarity

We will frequently need to recognize when two diagrams code the same set even in situations where the Mostowski collapse function is not available. If we are dealing with extensional, accessible pointed diagrams, then two such diagrams code the same set if and only if they are isomorphic. However our diagrams are typically neither, so we will instead use the notion of *bisimilarity* (see for example [1]). This is very well-behaved on well-founded graphs.

Definition 5.3 *A* bisimulation *between directed graphs G and H is a relation $\sim$ relating nodes of G to nodes of H, such that for all nodes u in G and v in H, $u \sim v$ holds if and only if both of the following hold:*

1. *For every predecessor u' of u in G, there is a predecessor v' of v in H such that $u' \sim v'$*
2. *For every predecessor v' of v in H, there is a predecessor u' of u in G such that $u' \sim v'$.*

Recall that a diagram $\langle a, E\rangle$ represents the directed graph with nodes $[a]$ and edges $[a]^2 \cap E$.

Definition 5.4 *We say that two diagrams $\langle a, E\rangle$ and $\langle b, F\rangle$ are* bisimilar *if there is a bisimulation $\sim$ between $\langle a, E\rangle$ and $\langle b, F\rangle$ such that $a \sim b$.*

Lemma 5.5 *There is at most one bisimulation between two diagrams.*

Proof Suppose $\sim_1$ and $\sim_2$ are distinct bisimulations between diagrams $\langle a, E\rangle$ and $\langle b, F\rangle$. Choose u $<$-minimal in $[a]$ such that there is a $v \in [b]$ for which $\sim_1$ and $\sim_2$ differ on the pair $\langle u, v\rangle$, and fix a $<$-minimal such v. Then $\sim_1$ and $\sim_2$ agree on all predecessors of u and v. Therefore, by the definition of a bisimulation, they must agree on $\langle u, v\rangle$, giving a contradiction. □

Lemma 5.6 *Given diagrams $\langle a, E\rangle$, $\langle b, F\rangle$, there is at least one bisimulation between them, and they are bisimilar if and only if $M(a, E) = M(b, F)$.*

Proof Define a relation $\sim$ by $u \sim v$ if and only if $M(u, E) = M(v, F)$. It follows directly from the definitions that this is a bisimulation between $\langle a, E\rangle$ and $\langle b, F\rangle$. We then apply Lemma 5.5. □

Lemma 5.7 *There is a* $\mathrm{CRSF}_{\subseteq}$ *function $B(a, E, b, F)$ computing the bisimulation between diagrams $\langle a, E\rangle$ and $\langle b, F\rangle$, by outputting the bisimulation as a set of ordered pairs.*

Proof First observe that $B(a, E, b, F) \subseteq [a] \times [b]$, so we can potentially define B directly by a subset-bounded recursion. We will use recursion on a. Given a set $S = \{\sim_c : c \in a\}$, where each $\sim_c$ is the bisimulation between $\langle c, E\rangle$ and $\langle b, F\rangle$ output by $B(c, E, b, F)$, let $\sim = \bigcup S$. Then by the uniqueness of bisimulations, $\sim \cap ([c] \times [b]) = \sim_c$ for every $c \in a$. It follows that $\sim$ has all the properties of a bisimulation between $\langle a, E\rangle$ and $\langle b, F\rangle$ except possibly at a, and we can extend $\sim$ to a by adding a pair $\langle a, y\rangle$ for every $y \in [b]$ which satisfies both conditions from Definition 5.3. □

5.3 Coding collections of sets

We use a generalized notion of a 0/1 matrix to allow a single set W to encode a collection of sets. The intuition is that the pairs $\langle c, x\rangle$ in W encode the Mostowski graph of the c-th set encoded by W.

Definition 5.8 *For sets W and c, we define the* c-th row of W*, denoted $W^{(c)}$, as the set $\{x : \langle c, x\rangle \in W\}$.*

Note that $W^{(c)} \subseteq \bigcup\bigcup W$, so the function $W, c \mapsto W^{(c)}$ is in $\mathrm{CRSF}_{\subseteq}$ by separation. The next lemma allows us to use such a matrix to code a set of sets.

Lemma 5.9 *The function $f(W, a, b) = \{M(a, W^{(c)}) : c \in b\}$ is in $\mathrm{CRSF}^{+}_{\subseteq}$.*

Proof Let $e = b\#a$. Our strategy is to construct from W a single embedded graph $\langle e, E\rangle$ such that $f(W, a, b) = M(e, E)$. We will write σ, π_1, π_2 for the functions $\sigma_{b,a}$, $\pi_{1,b,a}$, $\pi_{2,b,a}$. Define

$$E' = \{\langle u, v\rangle \in [e]^2 : \pi_1(u) = \pi_1(v) \neq b \wedge \langle \pi_2(u), \pi_2(v)\rangle \in W^{(\pi_1(u))}\}.$$

For each $c < b$ this puts the structure of $W^{(c)} \cap [a]^2$ onto copy c of the graph of a inside the graph of e, so that $M(\sigma(c, a), E') = M(a, W^{(c)})$. On the other hand $\sigma(b, a) = e$ is not connected to anything in E', so $M(e, E') = 0$. Let $E = E' \cup \{\langle \sigma(c, a), e\rangle : c \in b\}$. Then $M(e, E)$ equals $\{M(\sigma(c, a), E') : c \in b\}$, which is the required set. □

6 Boolean circuits

This section defines computations with unbounded fan-in Boolean circuits encoded by sets. We first introduce circuits that compute functions mapping strings to strings. We then extend this to circuits computing set functions, which operate on sets encoded as diagrams or strings. The set value that is computed by the circuit will be extracted from its output using the embedded Mostowski collapse.

Definition 6.1 *A* circuit *is a triple $\langle c, E, \lambda\rangle$ where $\langle c, E\rangle$ is a diagram and λ is a function from $[c]$ to the set of symbols $\{0, 1, *, \bigwedge, \bigvee, \neg\}$ (which we identify with the numbers $0, \ldots, 5$) such that each node labelled $\neg$ has exactly one E-predecessor. The nodes labelled $*$ are called* input nodes*. We say that the circuit has* size c*.*

Conjuctions and disjunctions may have arbitrary fan-in. The labels 0 and 1 represent the constant values *False* and *True*.

Definition 6.2 *Let $C = \langle c, E, \lambda \rangle$ be a circuit, and let a be its set of input nodes. Given any a-string A, a* computation of C on A *is a $[c]$-string W which, informally, assigns $0/1$ values to the nodes of C in such a way that each input node gets the same value that it has in A, and all other nodes get values according to the usual interpretations of their symbols in a circuit. That is, for all $u \in [c]$,*

1. *if $\lambda(u) = 0$ then $u \notin W$*
2. *if $\lambda(u) = 1$ then $u \in W$*
3. *if $\lambda(u) = *$ then $u \in W \leftrightarrow u \in A$*
4. *if $\lambda(u) = \bigwedge$ then $u \in W \leftrightarrow \forall v{<}u\, (\langle v, u \rangle \in E \rightarrow v \in W)$*
5. *if $\lambda(u) = \bigvee$ then $u \in W \leftrightarrow \exists v{<}u\, (\langle v, u \rangle \in E \wedge v \in W)$*
6. *if $\lambda(u) = \neg$ then $u \in W \leftrightarrow \exists v{<}u\, (\langle v, u \rangle \in E \wedge v \notin W)$.*

In general, to guarantee that a circuit has a computation it is enough for the graph underlying the circuit to be well-founded. However we need the extra condition that the graph is embedded in c for the next lemma.

Lemma 6.3 *There is a $\mathrm{CRSF}_{\subseteq}$ function which takes a circuit C and an a-string A, as in Definition 6.2, and outputs a computation of C on A.*

Proof Let $f(C, A, v)$ be the function that outputs a $[v]$-string assigning correct values to all nodes $u \leq v$ in the circuit. Since $f(C, A, v) \subseteq [v]$ it is possible to define f directly by subset-bounded recursion. Suppose we are given $S = \{f(C, A, u) : u \in v\}$. Then $W' = \bigcup S$ assigns the correct values to all nodes u of the circuit with $u < v$, and since $\langle c, E \rangle$ is a diagram, this includes all nodes such that $\langle u, v \rangle \in E$. Hence we have enough information to extend W' to a $[v]$-string W which also assigns the correct value to v, and clearly this can be done in $\mathrm{CRSF}_{\subseteq}$. □

We extend the definition to handle computations on tuples of strings. When c, $a_1, \ldots, a_k$ and p are finite ordinals, the following is equivalent to the usual definition of a finite Boolean circuit.

Definition 6.4 *A* circuit with input sizes $a_1, \ldots, a_k$ and output size p *is a tuple $\langle c, E, \lambda, \vec{a}, p, \mu, \nu \rangle$ where $\langle c, E, \lambda \rangle$ is a circuit and we are also given*

1. *A partial function* $\mu : [c] \to (\{0\} \times a_1) \cup \cdots \cup (\{k-1\} \times a_k)$ *which maps every input node to a member of (a disjoint copy of) some* a_i*, and*
2. *A function* $\nu : p \to [c]$ *which maps every element of* p *to some node in* $[c]$*. We call the range of* ν *the* output nodes*.*

Definition 6.5 *The* circuit evaluation function *takes inputs* $C, \vec{A}$ *where* $C = \langle c, E, \lambda, \vec{a}, p, \mu, \nu \rangle$ *is a circuit and* $\vec{A}$ *is a tuple of strings, with each* A_i *an* a_i*-string. It outputs the* p*-string* $C(\vec{A})$ *computed by* C *on inputs* $\vec{A}$*.*

Lemma 6.6 *The circuit evaluation function is in* $\mathrm{CRSF}_{\subseteq}$*.*

Proof We construct a computation W of the circuit as in the proof of Lemma 6.3. We recover the output by separation as $\{i \in p : \nu(i) \in W\}$. $\square$

Definition 6.7 *A family* $C_{\vec{a}}$ *of circuits parametrized by a tuple* $\vec{a}$ *of sets is* small *if there are smash-terms* $s, t, u_1, \ldots, u_k$ *such that the size of* $C_{\vec{a}}$ *is* $s(\vec{a})$*, and* $C_{\vec{a}}$ *has input sizes* $u_1(\vec{a}), \ldots, u_k(\vec{a})$ *and output size* $t(\vec{a})$*. We say that a family* $C_{\vec{a},\vec{b}}$ *is small* with size bounds independent of $\vec{b}$ *if the parameters* $\vec{b}$ *do not appear in any of these smash-terms.*

Consider a set function $f(x)$ which, when its inputs are sets embeddable in a, outputs a set embeddable in p. Using diagrams, we can code the inputs and output of f respectively as $[a]^2$-strings and $[p]^2$-strings. We can then ask whether the corresponding function from strings to strings is computed by a small circuit.

Definition 6.8 *Let* $f(x_1, \ldots, x_k)$ *be any set function and let* $a_1, \ldots, a_k$ *be a tuple of sets. Suppose that* C *is a circuit with input sizes* $[a_1]^2, \ldots, [a_k]^2$ *and output size* $[p]^2$ *for some* p*, such that for every tuple* $E_1, \ldots, E_k$ *of strings, with each* E_i *an* $[a_i]^2$*-string, we have*

$$M(p, C(E_1, \ldots, E_k)) \;=\; f(M(a_1, E_1), \ldots, M(a_k, E_k))$$

where $C(E_1, \ldots, E_k)$ *is the* $[p]^2$*-string as defined in Lemma 6.6. Then we say that* C computes f on sets embeddable in $a_1, \ldots, a_k$*.*

We say that f has small circuits *if there is a family* $C_{\vec{a}}$ *of small circuits such that, for all* $\vec{a}$*,* $C_{\vec{a}}$ *computes* f *on sets embeddable in* $\vec{a}$*.*

We next describe a class of simple formulas, which we will use both for constructing circuits and to define a notion of uniformity for circuits. Note that the terms in the language below are exactly the smash-terms.

Definition 6.9 *A $\Delta_0^\#$ formula is a formula in the language $\{\in, 0,$ pairing, tc, $\times$, $\odot$, $\#\}$ in which all quantifiers are bounded, of the form $\exists x{<}t$ or $\forall x{<}t$. We say that a family of sets is $\Delta_0^\#$-definable (with parameters $\vec{z}$) if there is a $\Delta_0^\#$ formula φ and a smash-term t such that the sets in the family have the form $\{u \in t(\vec{z}) : \varphi(u, \vec{z})\}$.*

Definition 6.10 *A family $C_{\vec{z}}$ of circuits is* $\Delta_0^\#$-uniform *if it is small and the sets $E \subseteq [c]^2$, $\lambda \subseteq [c] \times 6$, $\mu \subseteq [c] \times (\{0\} \times a_1) \cup \cdots \cup (\{k{-}1\} \times a_k))$ and $\nu \subseteq p \times [c]$ describing the circuit $C_{\vec{z}}$ are all $\Delta_0^\#$-definable with parameters $\vec{z}$.*

Below, to save space we will write e.g. "$a_1, \ldots, a_k$-strings $A_1, \ldots, A_k$" instead of "an a_1-string A_1, an a_2-string A_2, ..." etc.

The next lemma states that the truth-sets of certain kinds of $\Delta_0^\#$ formulas can be computed by $\Delta_0^\#$-uniform circuits. This is a crucial tool underlying the proof in Section 7 that $\mathrm{CRSF}^+_\subseteq$ has small circuits.

Lemma 6.11 *Let φ be a $\Delta_0^\#$ formula*

$$\varphi(x_1, \ldots, x_\ell, a_1, \ldots, a_m, b_1, \ldots, b_n, U_1, \ldots, U_k)$$

in which all occurrences of variables U_i are immediately to the right of an $\in$ symbol in an atomic formula of the form $t \in U_i$, and in which only variables a_i can appear in terms bounding quantifiers. It is not necessary that all the variables are present in φ. Let $t(\vec{a}), s_1(\vec{a}), \ldots, s_k(\vec{a})$ be smash-terms.

There is a $\Delta_0^\#$-uniform family of circuits $C_{\vec{a},\vec{b}}$ with size bounds independent of $\vec{b}$, with input sizes $s_1(\vec{a}), \ldots, s_k(\vec{a})$ and output size $t(\vec{a})$, such that for all $\vec{a}$ and $\vec{b}$ and all $s_1(\vec{a}), \ldots, s_k(\vec{a})$-strings $U_1, \ldots, U_k$ we have

$$C_{\vec{a},\vec{b}}(\vec{U}) = \{\langle \vec{x} \rangle \in t(\vec{a}) : \varphi(\vec{x}, \vec{a}, \vec{b}, \vec{U})\}.$$

Proof We fix m, n and k and use induction on the complexity of φ. Circuits will be given with descriptions that can easily be turned into $\Delta_0^\#$ formulas.

Suppose φ has the form $r_1(\vec{x}, \vec{a}, \vec{b}) \in r_2(\vec{x}, \vec{a}, \vec{b})$ for terms r_1, r_2. We compute the $t(\vec{a})$-string

$$\{\langle \vec{x} \rangle \in t(\vec{a}) : r_1(\vec{x}, \vec{a}, \vec{b}) \in r_2(\vec{x}, \vec{a}, \vec{b})\}$$

using a circuit of size $t(\vec{a})$. Each node $u \in [t(\vec{a})]$ of the circuit is labelled 1 if $u = \langle \vec{x} \rangle$ for some $\vec{x}$ with $r_1(\vec{x}, \vec{a}, \vec{b}) \in r_2(\vec{x}, \vec{a}, \vec{b})$, and is otherwise labelled 0 (there are no input nodes or nodes labelled with connectives). The set E of

edges is empty. The function ν mapping elements of the output size $t(\vec{a})$ to output nodes is just the identity.

The case where φ has the form $r(\vec{x}, \vec{a}, \vec{b}) \in U_i$ is broadly similar. This time each node $u \in [t(\vec{a})]$ is labelled with a $*$ (that is, as an input node) if $u = \langle \vec{x} \rangle$ for some $\vec{x}$ with $r(\vec{x}, \vec{a}, \vec{b}) \in s_i(\vec{a})$, and is otherwise labelled 0. The output is arranged as in the previous case. The function μ mapping input nodes to the disjoint union of input sizes maps $\langle \vec{x} \rangle$ to $\langle i{-}1, r(\vec{x}, \vec{a}, \vec{b}) \rangle$ (we use $i{-}1$ to match the notation in Definition 6.4), with the result that a computation W assigns 1 to $\langle \vec{x} \rangle$ if and only if $r(\vec{x}, \vec{a}, \vec{b}) \in U_i$.

Suppose φ has the form $\varphi_1 \wedge \varphi_2$. Let C_1 and C_2 be circuits for respectively

$$\{\langle \vec{x} \rangle \in t(\vec{a}) : \varphi_1(\vec{x}, \vec{a}, \vec{b}, \vec{U})\} \quad \text{and} \quad \{\langle \vec{x} \rangle \in t(\vec{a}) : \varphi_2(\vec{x}, \vec{a}, \vec{b}, \vec{U})\}$$

with sizes $c_1 = c_1(\vec{a})$ and $c_2 = c_2(\vec{a})$. We define a circuit D with size $d = t(\vec{a}) \odot \{c_2\} \odot \{c_1\}$. This means that the underlying graph of D consists of a copy of $\mathcal{G}(c_1)$ at the bottom, with a copy of $\mathcal{G}(c_2)$ above it, and a copy of $\mathcal{G}(t(\vec{a}))$ above that, with the sink node of each component connected to the source node of the next component. We will call the components C_1', C_2' and O. The labellings, edges and connections to inputs in C_1' are copied from C_1, and similarly for C_2' and C_2. So, for example, if C_2 has an edge $\langle u, v \rangle$ then D has an edge $\langle u \odot \{c_1\}, v \odot \{c_1\} \rangle$. The function ν maps every element of the output size $t(\vec{a})$ to the corresponding node of O, that is, $\nu : u \mapsto u \odot \{c_2\} \odot \{c_1\}$, and these nodes in the image of ν are labelled with $\bigwedge$. The other nodes of O are labelled with 0 and are not used. Finally, in the original circuits each $u \in t(\vec{a})$ was associated with an output node $\nu_1(u)$ in C_1 and an output node $\nu_2(u)$ in C_2. In D we connect each node $u \odot \{c_2\} \odot \{c_1\}$ labelled $\bigwedge$ in O to the nodes of D corresponding to $\nu_1(u)$ and $\nu_2(u)$, that is, to $\nu_1(u)$ in C_1' and $\nu_2(u) \odot \{c_1\}$ in C_2'.

Suppose φ has the form $\forall x' {\in} s(\vec{a})\, \psi(x', \vec{x}, \vec{a}, \vec{b}, \vec{U})$. Let C be a circuit for

$$\{\langle x', \vec{x} \rangle \in s(\vec{a}) \times t(\vec{a}) : \psi(x', \vec{x}, \vec{a}, \vec{b}, \vec{U})\}$$

with size $c = c(\vec{a})$. We construct a circuit D of size $t(\vec{a}) \odot \{c\}$, consisting of C with a copy O of $\mathcal{G}(t(\vec{a}))$ above it, similarly to the previous case. For the output nodes of D we take all nodes $u \odot \{c\}$ in O for $u \in t(\vec{a})$, and label them with $\bigwedge$. If such a node has the form $\langle \vec{x} \rangle \odot \{c\}$, we connect it by an edge to every output node $\nu_C(\langle x', \vec{x} \rangle)$ in C with $x' \in s(\vec{a})$, where ν_C is C's function mapping elements of its output size $s(\vec{a}) \times t(\vec{a})$ to its output nodes.

Negation is handled similarly. □

We finish this section by giving three concrete examples of families of small uniform circuits, all of which we will need in the next section. The

first example, union, is now an easy consequence of Lemma 6.11. The other two, circuits for computing bisimilarity and the ancestor relation, are more complicated.

Lemma 6.12 *The union function has $\Delta_0^\#$-uniform circuits.*

Proof Let $\langle a, E\rangle$ be a diagram. Let the $[a]^2$-string F be defined from the $[a]^2$-string E by

$$\langle x, y\rangle \in F \Leftrightarrow \begin{cases} \langle x, y\rangle \in E & \text{if } y \neq a \\ \exists z{\in}[a],\, \langle x, z\rangle \in E \ \wedge\ \langle z, a\rangle \in E & \text{if } y = a. \end{cases}$$

Then $M(a, F) = \bigcup M(a, E)$. The result follows by Lemma 6.11. □

For the next two examples we will need a technical lemma, to help construct circuits defined using smash.

Lemma 6.13 *For $x \in [a]$, $y \in [b]$ and $w \in [a\#b]$ the relation $\sigma_{a,b}(x, y) = w$ is definable by a $\Delta_0^\#$ formula which uses only terms in a and b as bounds on quantifiers.*

Proof We have $\sigma_{a,b}(x, y) = w$ if and only if $w = y \odot \{z\#b : z \in x\}$, which holds if and only if

$$\exists s{\in}[w],\, w = y \odot s \ \wedge\ s = \{z\#b : z \in x\}.$$

This is $\Delta_0^\#$-definable, and we can bound all quantifiers by $[a\#b] \cup [a]$. □

Lemma 6.14 *There is a family $C_{a,b}$ of $\Delta_0^\#$-uniform circuits, with input sizes $[a]^2$ and $[b]^2$ and with output size $[a] \times [b]$, which take an $[a]^2$-string E and a $[b]^2$-string F and output a string giving the bisimulation between diagrams $\langle a, E\rangle$ and $\langle b, F\rangle$.*

Proof We will imitate the recursion on a used to show that bisimilarity is in $\mathrm{CRSF}_\subseteq$ in Lemma 5.7. For $u \in [a]$ we write $\sim_u$ for the bisimulation between $\langle u, E\rangle$ and $\langle b, F\rangle$, which we treat as a $[a] \times [b]$-string.

Let $R_u = \bigcup\{\sim_{u'} : u' \in u\}$. Then for $\langle x, y\rangle \in [a]\times[b]$ we have $\langle x, y\rangle \in \sim_u$ if and only if either $\langle x, y\rangle \in R_u$, or $x = u$ and the conditions from Definition 5.3 hold, that is,

$$\begin{aligned} &\forall x'{\in}[a]\,\big(\langle x', x\rangle \in E \to \exists y'{\in}[b]\,(\langle y', y\rangle \in F \wedge \langle x', y'\rangle \in R_u)\big) \\ \wedge\ &\forall y'{\in}[b]\,\big(\langle y', y\rangle \in F \to \exists x'{\in}[a]\,(\langle x', x\rangle \in E \wedge \langle x', y'\rangle \in R_u)\big). \end{aligned}$$

Thus $\sim_u$ is expressed by a $\Delta_0^\#$ formula to which we can apply Lemma 6.11, giving $\Delta_0^\#$-uniform circuits $C_{a,b,u}$ computing $\sim_u$ from E, F and R_u, with size bounds independent of u. Let $t = t(a,b)$ be the size of $C_{a,b,u}$ and let λ_u, μ_u and ν_u be the functions describing respectively its internal, input and output gates.

Our circuit C computing bisimulations has size $a\#t$ and functions λ, μ and ν. It is formed as follows. For each $u \in [a]$:

1. C has an edge $\langle \sigma_{a,t}(u,x), \sigma_{a,t}(u,y)\rangle$ for each edge $\langle x, y\rangle$ in $C_{a,b,u}$.
2. For each $x \in [t]$, the internal gate $\lambda(\sigma_{a,t}(u,x))$ is the same as $\lambda_u(x)$.
3. For each $x \in [t]$, if $\mu_u(x) = \langle 0, z\rangle$ for $z \in [a]^2$ (representing an input from E) or if $\mu_u(x) = \langle 1, z\rangle$ for $z \in [b]^2$ (representing an input from F) then $\sigma_{a,t}(u,x)$ is an input node of C and $\mu(\sigma_{a,t}(u,x)) = \mu_u(x)$.

In other words via the map $x \mapsto \sigma_{a,t}(u,x)$ the u-th copy of t inside $a\#t$, which we will call C_u, is given the same internal structure as $C_{a,b,u}$ and accesses inputs from E and F in the same way. Then:

4. For each $x \in [t]$, if $\mu_u(x) = \langle 2, z\rangle$ for $z \in [a]\times[b]$ (representing an input from R_u) then $\lambda(\sigma_{a,t}(u,x)) = \bigvee$ and for each $u' \in u$ there is an edge in C connecting $\sigma_{a,t}(u,x)$ with the "output node" of $C_{u'}$ corresponding to z, that is, the node $\sigma_{a,t}(u', \nu_{u'}(z))$. (If $u = 0$, this is equivalent to setting $\lambda(\sigma_{a,t}(u,x)) = 0$.)
5. For each $z \in [a]\times[b]$ we set $\nu(z) = \sigma_{a,t}(a, \nu_a(z))$.

By item 4 the $[a]\times[b]$-string that the subcircuit C_u gets as its third "input" is the union of the $[a]\times[b]$-strings "output" by the subcircuits $C_{u'}$ for $u' \in u$. Thus by induction on u the "output" of C_u is $\sim_u$. Finally item 5 reads off the "output" of the top subcircuit C_a as the output of our circuit C. □

Lemma 6.15 *There is a family C_a of $\Delta_0^\#$-uniform circuits, with input and output size $[a]^2$, which take as input a string E and output a string giving the relation "x is an E-ancestor of y in the diagram $\langle a, E\rangle$".*

Proof Let us write $\lhd_u$ for the E-ancestor relation on $\langle u, E\rangle$. Let $R_u = \bigcup\{\lhd_v : v \in u\}$. Then for $x, y \in [u]$, we have $\langle x, y\rangle \in \lhd_u$ if and only if

$$x = y \vee \langle x, y\rangle \in R_u \vee \exists z{\in}[a]\,(z < y \wedge \langle x, z\rangle \in R_u \wedge \langle z, y\rangle \in E).$$

Hence by Lemma 6.11 there is a small, $\Delta_0^\#$-uniform circuit $C_{a,u}$ computing $\lhd_u$ from E and R_u, with size $t(a)$ independent of u.

Our circuit has size $a\#t(a)$ and is formed by taking $\mathcal{G}(a)$ and replacing each node u with a copy of $C_{a,u}$, adding edges so that $C_{a,u}$ gets as input E and the union of the outputs of $C_{a,v}$ for $v \in u$, exactly as in Lemma 6.14. □

7 Small circuits for $\mathrm{CRSF}^+_\subseteq$

Theorem 7.1 *Every* $\mathrm{CRSF}^+_\subseteq$ *function has* $\Delta_0^\#$*-uniform circuits.*

To prove this it is enough to show that all our initial functions have such circuits, and that the class of functions with such circuits is closed under composition and subset-bounded recursion.

Lemma 7.2 *The class of set functions with* $\Delta_0^\#$*-uniform circuits is closed under composition.*

Proof Let g be an m-ary set function computed by a circuit family $D_{\vec{x}}$ with size $d(\vec{x})$. Let $f_1, \ldots, f_m$ be n-ary set functions, with f_i computed by a circuit family $C^i_{\vec{y}}$ with size $c_i(\vec{y})$ and output size $[s_i(\vec{y})]^2$. Then the circuit to compute the composition of g and $\vec{f}$ on inputs embeddable in $\vec{y}$ has size

$$d(s_1(\vec{y}), \ldots, s_m(\vec{y})) \odot \{c_m(\vec{y})\} \odot \cdots \odot \{c_1(\vec{y})\}$$

and is formed in the obvious way, by connecting copies of $D_{s_1(\vec{y}),\ldots,s_m(\vec{y})}$ and $C^1_{\vec{y}}, \ldots, C^m_{\vec{y}}$ together. □

7.1 Initial functions

The projection function is trivial, and we dealt with the union function in Lemma 6.12. Pairing, conditional, set composition, set smash, transitive closure, cartesian product and embedded Mostowski collapse remain. In each case the proof uses Lemma 6.11.

Lemma 7.3 *The pairing function has* $\Delta_0^\#$*-uniform circuits.*

Proof Given diagrams $\langle a, E\rangle$ and $\langle b, F\rangle$ let $c = \{b\} \odot \{a\}$, so that $\mathcal{G}(c)$ has the structure of, first, a copy of $\mathcal{G}(a)$, then an edge connecting its sink to the source of a copy of $\mathcal{G}(b)$, then an edge connecting the sink of that to the global sink c. Let $\langle x, y\rangle \in G$ if and only if one of the following holds:

1. $\langle x, y\rangle \in E$
2. $\langle x, y\rangle = \langle x' \odot \{a\}, y' \odot \{a\}\rangle$ for some $\langle x', y'\rangle \in F$
3. $y = c$ and either $x = a$ or $x = b \odot \{a\}$.

By items 1 and 2, $M(a, G) = M(a, E)$ and $M(b \odot \{a\}, G) = M(b, F)$. Hence by item 3, $M(c, G) = \{M(a, E), M(b, F)\}$ as required. The lemma now follows from Lemma 6.11. □

Lemma 7.4 *(The characteristic functions of) membership and equality have $\Delta_0^\#$-uniform circuits.*

Proof This follows from Lemma 6.14. For example, to compute membership, given diagrams $\langle a, E\rangle$ and $\langle b, F\rangle$ we first construct a subcircuit computing the bisimulation $\sim$ between $\langle a, E\rangle$ and $\langle b, F\rangle$. Then $M(a, E) \in M(b, F)$ if and only if there exists $u < b$ with $\langle u, b\rangle \in F$ and $a \sim u$. □

Lemma 7.5 *The* $\mathrm{cond}_\in$ *function has $\Delta_0^\#$-uniform circuits.*

Proof Recall that the function $\mathrm{cond}_\in(e, f, g, h)$ takes the value e if $g \in h$ and takes the value f otherwise. Suppose we are given diagrams $\langle a, E\rangle$, $\langle b, F\rangle$, $\langle c, G\rangle$ and $\langle d, H\rangle$. We will output a diagram $\langle b \odot a, I\rangle$. Our circuit is formed from four subcircuits, which we will call W, X, Y and Z. These are combined in a similar way to Lemma 7.2.

The subcircuit W computes whether $M(c, G) \in M(d, H)$, as in Lemma 7.4.

The subcircuit X computes a $[b \odot a]^2$ string I_E with

$$\langle x, y\rangle \in I_E \Leftrightarrow \begin{cases} \langle x, y\rangle \in E & \text{if } y \neq a \\ \langle x, a\rangle \in E & \text{if } y = b \odot a \end{cases}$$

so that I_E has the structure of E but with the sink node moved to $b \odot a$, giving $M(b \odot a, I_E) = M(a, E)$.

The subcircuit Y computes a $[b \odot a]^2$ string I_F with

$$\langle x, y\rangle \in I_F \Leftrightarrow \langle x, y\rangle = \langle x' \odot a, y' \odot a\rangle \text{ for some } \langle x', y'\rangle \in F$$

so that $M(b \odot a, I_F) = M(b, F)$.

Finally, the subcircuit Z takes the outputs of W, X and Y, and outputs I_E if $M(c, G) \in M(d, H)$ and I_F otherwise. □

Lemma 7.6 *The set composition function $\odot$ has $\Delta_0^\#$-uniform circuits.*

Proof Suppose we are given diagrams $\langle a, E\rangle$ and $\langle b, F\rangle$. We will output a diagram $\langle a \odot b, G\rangle$, where $\langle x, y\rangle \in G$ if one of the following holds:

1. $\langle x, y\rangle \in F$
2. $\langle x, y\rangle = \langle b, y' \odot b\rangle$ for some $\langle x', y'\rangle \in E$ where x' is a source node of $\langle a, E\rangle$
3. $\langle x, y\rangle = \langle x' \odot b, y' \odot b\rangle$ for some $\langle x', y'\rangle \in E$ where x' is not a source node of $\langle a, E\rangle$

By item 1, $M(b, G) = M(b, F)$. Items 2 and 3 put the structure of $\langle a, E\rangle$ onto the copy of $\mathcal{G}(a)$ inside $\mathcal{G}(a \odot b)$, except that all source nodes of $\langle a, E\rangle$ get mapped to b. Hence $M(a \odot b, G) = M(a, E) \odot M(b, F)$. □

Lemma 7.7 *The set smash function # has $\Delta_0^{\#}$-uniform circuits.*

Proof Suppose we are given diagrams $\langle a, E\rangle$ and $\langle b, F\rangle$. We will output a diagram $\langle a\#b, G\rangle$, where $\langle x, y\rangle \in G$ if either of the following hold:

1. $\langle x, y\rangle = \langle \sigma_{a,b}(u, x'), \sigma_{a,b}(u, y')\rangle$ for some $u \in [a]$ and $\langle x', y'\rangle \in F$
2. $\langle x, y\rangle = \langle \sigma_{a,b}(u, b), \sigma_{a,b}(v, x)\rangle$ for some $\langle u, v\rangle \in E$ and some source node x of $\langle b, F\rangle$.

Here item 1 puts a copy of the structure of F onto each copy of $\mathcal{G}(b)$ inside $\mathcal{G}(a\#b)$, and item 2 connects the sources and sinks of these copies according to the edges of E. Hence $M(a\#b, G) = M(a, E)\#M(a, B)$. □

Lemma 7.8 *Transitive closure has $\Delta_0^{\#}$-uniform circuits.*

Proof We are given a diagram $\langle a, E\rangle$ and will output a diagram $\langle a, F\rangle$. We first use Lemma 6.15 to construct a subcircuit computing the E-ancestor relation on $\langle a, E\rangle$ as an $[a]^2$-string R. We then compute F as $E \cup \{\langle x, a\rangle : \langle x, a\rangle \in R\}$, so that $M(a, F) = \mathrm{tc}(M(a, E))$. □

Lemma 7.9 *Cartesian product has $\Delta_0^{\#}$-uniform circuits.*

Proof We are given diagrams $\langle a, E\rangle$ and $\langle b, F\rangle$. We will output a diagram $\langle c, G\rangle$ where $c = ((a \times b)\#2) \odot \{b\} \odot \{a\}$. We begin by copying the structures of $\langle a, E\rangle$ and $\langle b, F\rangle$ onto the copies of $\mathcal{G}(a)$ and $\mathcal{G}(b)$ inside $\mathcal{G}(c)$, so that $M(a, G) = M(a, E)$ and $M(b \odot \{a\}, G) = M(b, F)$.

Recall that ordered pairs are defined as $\langle e, f\rangle = \{\{e\}, \{e, f\}\}$. At the top of $\mathcal{G}(c)$, the graph $\mathcal{G}((a \times b)\#2)$ contains a disjoint copy of $\mathcal{G}(2)$ for each $\langle x, y\rangle \in a \times b$. Let

$$\tau : z \mapsto \sigma_{a\times b,2}(\langle x, y\rangle, z) \odot \{b\} \odot \{a\}$$

be the mapping from $\mathcal{G}(2)$ to this copy of $\mathcal{G}(2)$. If $\langle x, a\rangle \in E$ and $\langle y, b\rangle \in F$ then we add to G the edges

1. $\langle x, \tau(0)\rangle$, so that $M(\tau(0), G) = \{M(x, E)\}$
2. $\langle x, \tau(1)\rangle$ and $\langle y \odot \{a\}, \tau(1)\rangle$, so that $M(\tau(1), G) = \{M(x, E), M(y, F)\}$

3. $\langle \tau(0), \tau(2) \rangle$, $\langle \tau(1), \tau(2) \rangle$ and $\langle \tau(2), c \rangle$, so that $\langle M(x,E), M(y,F) \rangle \in M(c,G)$.

We do this for every $\langle x, y \rangle \in a \times b$. Hence $M(c,G) = M(a,E) \times M(y,F)$. $\square$

Lemma 7.10 *Embedded Mostowski collapse has $\Delta_0^{\#}$-uniform circuits.*

Proof We are given diagrams $\langle b, G_a \rangle$ and $\langle c, G_E \rangle$. We will output a diagram $\langle b, H \rangle$ such that if we let $a = M(b, G_a)$ and $E = M(c, G_E)$ then

$$M(b,H) = M(a,E).$$

For any set x, define $\kappa(x) = M(x, G_a)$, so that $\kappa(b) = a$. We put

$$H = \{\langle x, y \rangle \in [b]^2 : x < y \wedge \kappa(x) < \kappa(y) \wedge \langle \kappa(x), \kappa(y) \rangle \in E\}.$$

Before showing how to compute H with a circuit, we prove that $M(b,H) = M(a,E)$. We will show by $\in$-induction on y that $M(y,H) = M(\kappa(y), E)$ for all $y \in [b]$. Suppose this is true for all $x < y$. Then

$$\begin{aligned} M(y,H) &= \{M(x,H) : x < y \wedge \langle x, y \rangle \in H\} \\ &= \{M(\kappa(x), E) : x < y \wedge \kappa(x) < \kappa(y) \wedge \langle \kappa(x), \kappa(y) \rangle \in E\} \\ &= \{M(z,E) : z < \kappa(y) \wedge \langle z, \kappa(y) \rangle \in E\} \\ &= M(\kappa(y), E). \end{aligned}$$

Here the first and fourth equalities are the definition of M, and the second follows from the inductive hypothesis and the definition of H. One direction of the third equality follows from setting $z = \kappa(x)$. For the other direction, let $z < \kappa(y)$ and $\langle z, \kappa(y) \rangle \in E$. Since $z < \kappa(y)$ there is a finite sequence $z_1, \ldots, z_k$ such that $z = z_1 \in \cdots \in z_k \in \kappa(y)$. By the definition of κ, there is some $x_k < y$ such that $z_k = \kappa(x_k)$. Similarly we can find $x_{k-1} < x_k$ such that $z_{k-1} = \kappa(x_{k-1})$, and so on until we find $x_1 < x_2$ with $z = z_1 = \kappa(x_1)$. We put $x = x_1$.

To compute H with a circuit, we first construct subcircuits computing the bisimulation $\sim_a$ between $\langle b, G_a \rangle$ and $\langle b, G_a \rangle$; the bisimulation $\sim_{a,E}$ between $\langle b, G_a \rangle$ and $\langle c, G_E \rangle$; the bisimulation $\sim_E$ between $\langle c, G_E \rangle$ and $\langle c, G_E \rangle$; and the G_a-ancestor relation $\lhd_a$ on $\langle b, G_a \rangle$. Then for $x, y \in [b]$,

$$\begin{aligned} \kappa(x) < \kappa(y) &\Leftrightarrow M(x, G_a) \in \mathrm{tc}(M(y, G_a)) \\ &\Leftrightarrow \exists u{<}y,\ \ x \sim_a u \wedge u \lhd_a y. \end{aligned}$$

On the other hand, $\langle \kappa(x), \kappa(y) \rangle \in E$ if and only if

$$\exists x', y' < c,\ \ M(x, G_a) = M(x', G_E) \wedge M(y, G_a) = M(y', G_E) \\ \wedge \langle M(x', G_E), M(y', G_E) \rangle \in M(c, G_E)$$

which is equivalent to

$$\exists x', y' < c,\ x \sim_{a,E} x' \wedge y \sim_{a,E} y' \wedge \text{“}\langle c, G_E \rangle \vDash \langle x', y' \rangle \in c\text{”}.$$

where the expression in quotation marks means that we interpret $\langle x', y' \rangle \in c$ in the universe $[c]$ with the membership relation given by G_E and equality given by $\sim_E$. This can be written as a $\Delta_0^\#$ formula. □

7.2 Closure under recursion

Lemma 7.14 below establishes closure under subset-bounded recursion. We will need a few gadgets for the proof.

Lemma 7.11 *The function $x, y \mapsto x \cap y$ has $\Delta_0^\#$-uniform circuits. Furthermore we may assume that the circuit $C_{a,b}$ computing it on sets embeddable in a, b outputs an $[a]^2$-string E for a diagram $\langle a, E \rangle$.*

Proof We are given diagrams $\langle a, X \rangle$ and $\langle b, Y \rangle$. We first build a subcircuit to compute the bisimulation $\sim$ between them, then put

$$\langle i, j \rangle \in E \Leftrightarrow \begin{cases} \langle i, j \rangle \in X & \text{if } j \neq a \\ \langle i, j \rangle \in X \wedge \exists i' \in [b],\ \langle i', b \rangle \in Y \wedge i \sim i' & \text{if } j = a. \end{cases}$$ □

Lemma 7.12 *There is a $\Delta_0^\#$-uniform family $C_{a,u}$ of circuits with size bounds independent of u which, for $u \leq a$, take as input an $[a]^2$-string X and output an $[a]^2$-string E such that $M(a, E) = M(u, X)$.*

Proof If $u = a$ we output X. Otherwise we take X, remove all edges $\langle i, a \rangle$, then add an edge $\langle i, a \rangle$ for every $\langle i, u \rangle \in X$, and output the result. □

Lemma 7.13 *There is a $\Delta_0^\#$-uniform family $C_{a,b,u}$ of circuits with size bounds independent of u which, for $u \leq a$, take as input an $[a] \times [b]^2$-string W and an $[a]^2$-string X, and output an $[a \# b]^2$ string E such that*

$$M(a \# b, E) = \{M(b, W^{(v)}) : v < u \wedge \langle v, u \rangle \in X\}.$$

Proof We repeat the construction from the proof of Lemma 5.9. We will write σ, π_1, π_2 for the functions $\sigma_{a,b}$, $\pi_{1,a,b}$, $\pi_{2,a,b}$. Define

$$E' = \{\langle i, j\rangle \in [a\#b]^2 : \pi_1(i) = \pi_1(j) \neq a \wedge \langle \pi_2(i), \pi_2(j)\rangle \in W^{(\pi_1(i))}\}.$$

For each $v < a$ this puts the structure of $W^{(v)} \cap [b]^2$ onto copy v of the graph of b inside the graph of E', so that $M(\sigma(v, b), E') = M(b, W^{(v)})$. On the other hand $\sigma(b, a) = a\#b$ is not connected to anything in E', so $M(a\#b, E') = 0$. To rectify this, let $E = E' \cup \{\langle \sigma(v, b), a\#b\rangle : v < u \ \wedge \langle v, u\rangle \in X\}$. □

We can now prove the main result of this subsection.

Lemma 7.14 *The class of functions with $\Delta_0^\#$-uniform circuits is closed under subset-bounded recursion.*

Proof Suppose that g and h are set functions with $\Delta_0^\#$-uniform circuits. Let f be the function satisfying

$$f(x, z) = g(\{f(y, z) : y \in x\}, x, z) \cap h(x, z)$$

where for simplicity we consider only a single parameter z rather than a tuple of parameters $\vec{z}$ (this does not change anything important). We are given sets a, c and must construct a circuit computing $f(x, z)$ on all sets x, z embeddable in respectively a, c. Consider arbitrary input strings X, Z and let $x = M(a, X)$ and $z = M(c, Z)$.

We would like to build a circuit similar to those for Lemmas 6.14 and 6.15, in which we put together many copies of the circuits for g and h to simulate computing $f(x, z)$ by recursion on x. However we are given X as a string input, and cannot use it as a parameter when constructing our circuit. Instead we will model a recursion on a.

Let $[t]^2 = [t(a, c)]^2$ be the output size of the given circuit computing $h(x, z)$ on sets embeddable in a, c. We will build a circuit which, as it computes, constructs for each node $u \in [a]$ a $[t]^2$-string F_u such that

$$M(t, F_u) = f(M(u, X), z) \tag{1}$$

and hence in particular $M(t, F_a) = f(x, z)$.

Let $g'(s, x, z) = g(s, x, z) \cap h(x, z)$ and let $C = C_{a,c}$ be a circuit computing g' on sets s, x, z embedded in $a\#t, a, c$. By our assumptions and Lemma 7.11, we may assume that C is $\Delta_0^\#$-uniform and furthermore that the output of

C is a $[t]^2$-string. Let $D_u = D_{a,u}$ be the circuit from Lemma 7.12 with the property that

$$M(a, D_u(X)) = M(u, X)$$

for $u \leq a$. Let $E_u = E_{a,t,u}$ be the circuit from Lemma 7.13 with the property that, for any $[a] \times [t]^2$-string W,

$$M(a\#t, E_u(W, X)) = \{M(t, W^{(v)}) : v < u,\ \langle v, u\rangle \in X\}.$$

Combining these, let G_u be the circuit which takes an $[a] \times [t]^2$-string W, an $[a]^2$-string X and a $[c]^2$-string Z, and outputs the $[t]^2$-string

$$G_u(W, X, Z) = C(E_u(W, X), D_u(X), Z).$$

Suppose we have found F_v satisfying (1) for all $v < u$. If we define

$$W = \{\langle v, i, j\rangle \in [a] \times [t]^2 : v < u,\ \langle i, j\rangle \in F_v\}$$

then $W^{(v)} = F_v$ for all $v < u$, and if we let $F_u = G_u(W, X, Z)$ we have

$$\begin{aligned} M(t, F_u) &= M(t, C(E_u(W, X), D_u(X), Z)) \\ &= g'(M(a\#t, E_u(W, X)), M(a, D_u(X)), M(c, Z)) \\ &= g'(\{M(t, F_v) : v < u,\ \langle v, u\rangle \in X\}, M(u, X), z) \\ &= g'(\{f(y, z) : y \in M(u, X)\}, M(u, X), z) \\ &= f(M(u, X), z). \end{aligned}$$

We can now describe a circuit computing F_a. Its overall shape is similar to the circuit for bisimilarity in Lemma 6.14. We start by taking a copy of G_u for each $u \in [a]$, so the size of our circuit is $a\#q$ where q is the size of G_u (which is independent of u). We let each G_u take its inputs X and Z from the global inputs X and Z. Its input nodes for W correspond to triples $\langle v, i, j\rangle \in [a] \times [t]^2$. For each such node, if $v \not< u$ we relabel it with the constant 0. Otherwise we wire it to the output node of G_v corresponding to $\langle i, j\rangle$. In this way each G_u gets as input exactly the string W described above. Hence the output of the topmost subcircuit G_a is F_a. □

8 Consequences of small circuits

In this section we use our results about small circuits first to sharpen our characterization of $\mathrm{CRSF}^+_\subseteq$, and then to prove some lower bounds for it.

Theorem 8.1 *The* $\mathrm{CRSF}^+_\subseteq$ *functions are exactly the set functions with* $\Delta^\#_0$*-uniform circuits.*

Proof One direction is given by Theorem 7.1. It remains to show that every function with $\Delta^\#_0$-uniform circuits is in $\mathrm{CRSF}^+_\subseteq$. So suppose that $f(x_1, \ldots, x_k)$ is such a function. This means that there is such a family $C_{\vec{a}}$ of circuits with the property that, for all $a_1, \ldots, a_k$, the circuit $C_{\vec{a}}$ computes f on sets embeddable in $\vec{a}$, and that if $C_{\vec{a}} = \langle c, E, \lambda, \vec{s}, p, \mu, \nu\rangle$ then c and p are given by smash-terms in $\vec{a}$, each input size s_i is $[a_i]^2$, and E, λ, μ and ν are $\Delta^\#_0$-definable from $\vec{a}$.

To compute f on $\vec{a}$ in $\mathrm{CRSF}^+_\subseteq$ we first construct the circuit $C_{\vec{a}}$. This can be done in $\mathrm{CRSF}_\subseteq$, since by the uniformity conditions and the closure properties of $\mathrm{CRSF}_\subseteq$ there are $\mathrm{CRSF}_\subseteq$ functions computing each component of $C_{\vec{a}}$ from $\vec{a}$, and we can construct the usual ordered 7-tuples in $\mathrm{CRSF}_\subseteq$. Then for each i we can trivially construct from a_i the $[a_i]^2$-string $E_i := \in\restriction[a_i]$, so that $M(a_i, E_i) = a_i$. By Lemma 6.6, we can evaluate $C_{\vec{a}}$ on these strings with a $\mathrm{CRSF}_\subseteq$ function; then, with a $\mathrm{CRSF}^+_\subseteq$ function, we can output $M(t, C_{\vec{a}}(E_1, \ldots, E_k))$, where $p = t \times t$ is the output size of $C_{\vec{a}}$. □

It follows from the proof above that any function in $\mathrm{CRSF}^+_\subseteq$ can be computed using only a single Mostowski collapse at the end of the computation.

Theorem 8.2 *Let* $f \in \mathrm{CRSF}^+_\subseteq$*. Then there is a function* $g \in \mathrm{CRSF}_\subseteq$ *and a smash-term* t *such that* $f(\vec{a}) = M(t(\vec{a}), g(\vec{a}))$*.*

Proof By Theorem 7.1, the function f has $\Delta^\#_0$-uniform circuits. We now repeat the proof of Theorem 8.1, skipping the final step, so that g computes $C_{\vec{a}}(E_1, \ldots, E_k)$. □

Corollary 8.3 *The* $\mathrm{CRSF}^+_\subseteq$ *and* $\mathrm{CRSF}_\subseteq$ *relations are the same. Furthermore for every* $f \in \mathrm{CRSF}^+_\subseteq$ *the function* $g(\vec{a}, b) := f(\vec{a}) \cap b$ *is in* $\mathrm{CRSF}_\subseteq$*.*

Proof For the first part, observe that given a diagram $\langle t, E\rangle$ we can easily compute in $\mathrm{CRSF}_\subseteq$ whether $M(t, E) = 0$. For the second part, we first compute a diagram $\langle t, E\rangle$ with $M(t, E) = f(\vec{a})$, and then compute the bisimulation $\sim$ between $\langle t, E\rangle$ and the Mostowski graph of b. Then $f(\vec{a}) \cap b = \{c \in b : \exists u{<}t,\ \langle u, t\rangle \in E \wedge u \sim c\}$. □

We can generalize Theorem 8.2 into a useful result about computing codes for $\mathrm{CRSF}^+_\subseteq$ functions in $\mathrm{CRSF}_\subseteq$.

Lemma 8.4 *Let $f \in \mathrm{CRSF}^+_\subseteq$. Then there is a function $g \in \mathrm{CRSF}_\subseteq$ and a smash-term t such that $f(\vec{b}) = M(t(\vec{a}), g(\vec{a}, \vec{b}))$ whenever $\vec{a}$ and $\vec{b}$ are tuples of sets with $b_i \leq a_i$ for all i.*

Proof As before we repeat the proof of Theorem 8.1, except, rather than choosing E_i such that $M(a_i, E_i) = a_i$ we choose it so that $M(a_i, E_i) = b_i$. This can be done by first constructing a $[b_i]^2$-string E'_i such that $M(b_i, E'_i) = b_i$ and then using the method of Lemma 7.12. □

Corollary 8.5 $\mathrm{CRSF}^+_\subseteq$ *is closed under the replacement and union schemes. That is, for any* $\mathrm{CRSF}^+_\subseteq$ *function g, there are* $\mathrm{CRSF}^+_\subseteq$ *functions f and u with*

$$f(\vec{a}, c) = \{g(\vec{a}, b) : b \in c\} \quad \textit{and} \quad u(\vec{a}, c) = \bigcup_{b \in c} g(\vec{a}, b).$$

Proof By Lemma 8.4 there is a smash term t and a $\mathrm{CRSF}_\subseteq$ function h such that $h(\vec{a}, c, b) \subseteq [t(\vec{a}, c)]^2$ and $g(\vec{a}, b) = M(t(\vec{a}, c), h(\vec{a}, c, b))$ for every $b \in c$. We can now use the method of Section 5.3 for coding collections of sets. Let

$$W = \bigcup_{b \in c} \{b\} \times h(\vec{a}, c, b).$$

Then $M(t(\vec{a}, c), W^{(b)}) = g(\vec{a}, b)$ for every $b \in c$, and W is computable in $\mathrm{CRSF}_\subseteq$ from $\vec{a}$ and c using separation, since $W \subseteq c \times [t(\vec{a}, c)]^2$. Replacement follows by Lemma 5.9, and union follows immediately from replacement. □

The *course-of-values* of a function $f(x)$ on a set a (where f may possibly have other parameters) is defined in [4] as the set $\{\langle b, f(b)\rangle : b \in \mathrm{tc}(a)\}$. We will use a slightly different definition, in the spirit of Section 5.3, which is more convenient to use with subset-bounded recursion.

Definition 8.6 *We define $f_{\restriction c}(\vec{a}, -) := \bigcup_{b \in c} \{b\} \times f(\vec{a}, b)$, so that for $b \in c$ we have $f(\vec{a}, b) = f_{\restriction c}(\vec{a}, -)^{(b)}$.*

We define the course-of-values of $f(\vec{a}, x)$ on c *(with respect to the argument x) as $f_{\restriction \mathrm{tc}(c)}(\vec{a}, -)$, and will write this as $f_{<c}(\vec{a}, -)$.*

Definition 8.7 *The function f is obtained from g by* subset-bounded course-of-values recursion *with bound h if*

$$f(\vec{a}, b) = g(\vec{a}, b, f_{<b}(\vec{a}, -)) \cap h(\vec{a}, b).$$

Corollary 8.8 $\mathrm{CRSF}^+_\subseteq$ *is closed under subset-bounded course-of-values recursion.*

Proof Suppose $f(\vec{a},b) = g(\vec{a},b,f_{<b}(\vec{a},-)) \cap h(\vec{a},b)$ with $g,h \in \mathrm{CRSF}^+_\subseteq$. Define $F(\vec{a},b) = f_{\restriction[b]}(\vec{a},-)$. Then $F(\vec{a},b) \subseteq [b] \times \bigcup_{c\leq b} h(\vec{a},c)$, since each $f(\vec{a},c) \subseteq h(\vec{a},c)$. By Corollary 8.5 this bound on F is in $\mathrm{CRSF}^+_\subseteq$. Hence we may potentially define F by subset-bounded recursion. This is straightforward, as given $S = \{F(\vec{a},c) : c \in b\}$ we have that $\bigcup S = f_{<b}(\vec{a},-)$, so $f(\vec{a},b) = g(\vec{a},b,\bigcup S)$ and $F(\vec{a},b) = \bigcup S \cup (\{b\} \times f(\vec{a},b))$. Hence F is in $\mathrm{CRSF}^+_\subseteq$, and it follows immediately that f is as well. □

An interesting consequence of our characterization of $\mathrm{CRSF}^+_\subseteq$ in terms of circuits is that it allows us to use known circuit lower bounds to prove that certain functions are not in $\mathrm{CRSF}^+_\subseteq$.

Theorem 8.9 *There is no* $\mathrm{CRSF}^+_\subseteq$ *function* f *which computes the parity of* $|x|$ *for every hereditarily finite set* x*.*

Proof Suppose f is such a function. Then by Theorem 7.1 there is a smash-term t such that for every set a there is a circuit C_a of size $t(a)$ computing f on all sets embeddable in a. Choose a large $n \in \omega$ of the form 2^{2^k}. Let $a = \mathbb{P}(\mathbb{P}(k))$. Then $\mathrm{rank}(a) = k+2$, $|a| = n$ and $|[a]| \leq 2n$. Therefore the underlying graph $\mathcal{G}(t(a))$ of C_a has depth (that is, rank) polynomial in k and size polynomial in n. But since C_a computes the parity of all subsets of a, it is straightforward to build from it a circuit of depth polynomial in k and size polynomial in n which computes the parity of all n-bit strings, which is impossible by [8]. □

In contrast, by the simulation of polynomial time in $\mathrm{CRSF}^+_\subseteq$ in Section 4 there is a $\mathrm{CRSF}_\subseteq$ function which can compute the parity of a hereditarily finite set x, as long as x is a set of ordinals.

As a corollary of Theorem 8.9 we get a version of P $\neq$ NP for $\mathrm{CRSF}^+_\subseteq$. We first define a natural notion of NP for $\mathrm{CRSF}^+_\subseteq$.

Definition 8.10 *A* $\Sigma^\subseteq_1$-$\mathrm{CRSF}^+_\subseteq$ relation *is one of the form* $\exists y{\subseteq}t(x)\, \varphi(x,y)$*, where* $\varphi(x,y)$ *is a* $\mathrm{CRSF}^+_\subseteq$ *relation and* t *is a smash term.*

Corollary 8.11 *The relation* $\varphi(x)$ *expressing that* x *can be partitioned into a collection of unordered pairs is* $\Sigma^\subseteq_1$-$\mathrm{CRSF}^+_\subseteq$*, but is not equivalent to any* $\mathrm{CRSF}^+_\subseteq$ *relation, even on hereditarily finite sets* x*.*

That $\Sigma^\subseteq_1$-$\mathrm{CRSF}^+_\subseteq$ is different from $\mathrm{CRSF}^+_\subseteq$ on $\{0,1\}^\omega$ already follows from [12]. Say that a set $L \subseteq \{0,1\}^\omega$ is in NP if there is a set $M \subseteq \{0,1\}^\omega$

in P (that is, decided by a polynomial time infinite-time Turing machine) such that $x \in L \leftrightarrow \exists y \in \{0,1\}^\omega\, x \oplus y \in M$, where $x \oplus y \in \{0,1\}^\omega$ alternates bits from x with bits from y. Then [12] shows that every set in P is Borel, and gives an example of a set Δ in NP which is not Borel. By Theorems 4.5 and 4.6, P coincides with $\mathrm{CRSF}^+_\subseteq$ on $\{0,1\}^\omega$. It follows that Δ is in $\Sigma^{\subseteq}_1$-$\mathrm{CRSF}^+_\subseteq$ but not in $\mathrm{CRSF}^+_\subseteq$.

As was already discussed, since the above separation is based on parity, it has no relevance for whether the ordinary versions of P and NP are distinct.

9 Embedding-bounded recursion

We will write $\mathrm{CRSF}_\preccurlyeq$ for the class of CRSF functions in the sense of [4]. The goal of this section is to show that $\mathrm{CRSF}^+_\subseteq$ and $\mathrm{CRSF}_\preccurlyeq$ are the same. We recall some definitions from [4].

Definition 9.1 *A function τ is an* embedding of a into b, *written $\tau : a \preccurlyeq b$, if $\tau : \mathrm{tc}(a) \to \mathcal{P}(\mathrm{tc}(b))$ and for all $x, y \in \mathrm{tc}(a)$,*

1. *$\tau(x) \neq 0$*
2. *if $x \neq y$ then $\tau(x) \cap \tau(y) = 0$*
3. *if $x \in y$ then for every $v \in \tau(y)$, there is some $u \in \tau(x)$ with $u < v$.*

Definition 9.2 *Let $g(\vec{a}, b, x)$, $h(\vec{a}, b)$ and $\tau(x, \vec{a}, b)$ be functions. The function $f(\vec{a}, b)$ is obtained from $g(\vec{a}, b, x)$ by* embedding-bounded recursion *with bound $h(\vec{a}, b)$ and embedding function $\tau(x, \vec{a}, b)$ if*

$$f(\vec{a}, b) = g(\vec{a}, b, \{f(\vec{a}, c) : c \in b\})$$

and if for all $\vec{a}, b$ we have $\tau(x, \vec{a}, b) : f(\vec{a}, b) \preccurlyeq h(\vec{a}, b)$. The last condition means that the function $x \mapsto \tau(x, \vec{a}, b)$ is an embedding $f(\vec{a}, b) \preccurlyeq h(\vec{a}, b)$.

Definition 9.3 $\mathrm{CRSF}_\preccurlyeq$ *is the closure of the empty set, projections, pairing, union,* $\mathrm{cond}_\in$ *and set smash # functions under composition and embedding-bounded recursion.*

We first show that the two natural notions of embeddability coincide.

Lemma 9.4 *The following are equivalent:*

1. *There is a set E such that $a = M(b, E)$.*

2. There is a function τ such that $\tau : a \preccurlyeq b$.

Proof First suppose that $a = M(b, E)$. For $x < a$ define $\tau(x) = \{u < b : x = M(u, E)\}$. It is straightforward to show that τ has the properties of an embedding function. In particular, if $v \in \tau(y)$ and $x \in y$ then $x \in M(v, E)$ so, by the definition of embedded Mostowski collapse, $x = M(u, E)$ for some $u < v$, giving $u \in \tau(x)$ as required.

For the other direction, suppose $\tau : a \preccurlyeq b$. We first extend the domain of τ from $\mathrm{tc}(a)$ to $[a]$ by defining $\tau(a) = \{b\}$. It is easy to see that the three properties of an embedding function still hold. Now define E as the set of pairs $\langle u, v\rangle \in [b]^2$ such that

$$u < v \wedge \exists x, y{\in}[a],\ x \in y \wedge u \in \tau(x) \wedge v \in \tau(y).$$

We will prove by $\in$-induction on y that, for every $y \in [a]$ and $v \in \tau(y)$, $y = M(v, E)$. Since $\tau(a) = \{b\}$ it will follow that $a = M(b, E)$.

Fix $v \in \tau(y)$. To show $M(v, E) \subseteq y$, let $z \in M(v, E)$. Then $z = M(u, E)$ for some $u < v$ with $\langle u, v\rangle \in E$. By the definition of E and the properties of τ, there exists $x \in y$ with $u \in \tau(x)$. By the inductive hypothesis, $x = M(u, E)$. Therefore $x = z$, so $z \in y$.

To show $y \subseteq M(v, E)$, suppose $x \in y$. Since $v \in \tau(y)$, by the properties of τ there is some $u \in \tau(x)$ with $u < v$. Then $\langle u, v\rangle \in E$ and also by the inductive hypothesis $x = M(u, E)$. Therefore $x \in M(v, E)$. □

Theorem 9.5 $\mathrm{CRSF}^+_{\subseteq}$ *is contained in* $\mathrm{CRSF}_{\preccurlyeq}$.

Proof It is shown in [4] that $\mathrm{CRSF}_{\preccurlyeq}$ contains all the initial functions of $\mathrm{CRSF}_{\subseteq}$ and is closed under subset-bounded recursion. That is, all $\mathrm{CRSF}_{\subseteq}$ functions are in $\mathrm{CRSF}_{\preccurlyeq}$. So it is enough to show that embedded Mostowki collapse is in $\mathrm{CRSF}_{\preccurlyeq}$. We know from [4] that $\mathrm{CRSF}_{\preccurlyeq}$ is closed under *course-of-values embedding-bounded recursion*, which is a version of embedding-bounded recursion in which we are allowed us to use the sequence $\{\langle c, f(c, \vec{z})\rangle : c < b\}$ of all earlier values of f when calculating $f(b, \vec{z})$. We have

$$M(a, E) = \{M(b, E) : b < a \wedge \langle b, a\rangle \in E\}.$$

If $S = \{\langle b, M(b, E)\rangle : b < a\}$ is the sequence of earlier values of M then

$$M(a, E) = \{y \in \bigcup\bigcup S : \exists b{<}a,\ \langle b, y\rangle \in S \wedge \langle b, a\rangle \in E\}$$

which is a $\mathrm{CRSF}_{\preccurlyeq}$ function of S, E and a.

For this to be a valid instance of course-of-values embedding-bounded recursion, we also need to provide a $\mathrm{CRSF}_{\preccurlyeq}$ function embedding $M(a, E)$ into an existing $\mathrm{CRSF}_{\preccurlyeq}$ function of a and E. By the proof of Lemma 9.4, if $b = M(a, E)$ and we define $\tau(x) = \{u < a : x = M(u, E)\}$ then $\tau : b \preccurlyeq a$. To compute τ by a $\mathrm{CRSF}_{\preccurlyeq}$ function which does not use M, we first compute the bisimulation $\sim$ between the Mostowski graph of x and $\langle a, E\rangle$ (by Lemma 5.7 we can do this in $\mathrm{CRSF}_{\subseteq}$, and hence in $\mathrm{CRSF}_{\preccurlyeq}$) and then take $\tau(x) = \{u < a : x \sim u\}$. □

Theorem 9.6 $\mathrm{CRSF}_{\preccurlyeq}$ *is contained in* $\mathrm{CRSF}^{+}_{\subseteq}$.

Proof By Theorem 21 of [4], it is enough to show that $\mathrm{CRSF}^{+}_{\subseteq}$ is closed under $\preccurlyeq$-bounded recursion where the bounding term h is a #-term. Here a *#-term* is a stricter version of our smash-term: it is built only from variables, the constant 1, and the function symbols $\odot$ and #. So suppose that g and τ are $\mathrm{CRSF}^{+}_{\subseteq}$ functions, h is a #-term and f is a function such that for all $\vec{a}, b$

$$f(\vec{a}, b) = g(\vec{a}, b, \{f(\vec{a}, c) : c \in b\}) \quad \text{and} \quad \tau(x, \vec{a}, b) : f(\vec{a}, b) \preccurlyeq h(\vec{a}, b).$$

We must show that $f \in \mathrm{CRSF}^{+}_{\subseteq}$.

By Lemma 9.7 below we may assume that there is a function τ' in $\mathrm{CRSF}^{+}_{\subseteq}$ such that, for all $\vec{a}, b, d$ with $b \leq d$,

$$\tau'(x, \vec{a}, b, d) : f(\vec{a}, b) \preccurlyeq h(\vec{a}, d).$$

We will use τ' rather than τ because it is convenient to have one fixed bounding set throughout the recursion. Below we write h for $h(\vec{a}, d)$.

We will show $f \in \mathrm{CRSF}^{+}_{\subseteq}$ by defining a function $F(\vec{a}, b, d)$ in $\mathrm{CRSF}^{+}_{\subseteq}$ with the property that $F(\vec{a}, b, d) = \{b\} \times E$ for some $[h]^2$-string E such that $M(h, E) = f(\vec{a}, b)$ whenever $b \leq d$ (the $\{b\}$ is there to help us carry out a kind of course-of-values recursion). We will construct F using $\subseteq$-bounded recursion on b — note that $F(\vec{a}, b, d) \subseteq \{b\} \times [h]^2$. We then obtain $f(\vec{a}, b)$ as $M(h(\vec{a}, b), F(\vec{a}, b, b)^{(b)})$.

Consider the point in the recursion where we reach a set $b \leq d$. We are given the set $s = \{F(\vec{a}, c, d) : c \in b\}$. Let $W = \bigcup s$. By Lemma 5.9 we can compute $s' = \{M(h, W^{(c)}) : c \in b\}$. By the properties of F, each $W^{(c)}$ is a set E with $M(h, E) = f(\vec{a}, c)$. Therefore $s' = \{f(\vec{a}, c) : c \in b\}$.

We can now compute $f(\vec{a}, b)$ as $g(\vec{a}, b, s')$. Since we have a $\mathrm{CRSF}^{+}_{\subseteq}$ embedding $\tau' : f(\vec{a}, b) \preccurlyeq h$, we can use the construction in the proof of Lemma 9.4 to build a set $E \subseteq [h]^2$ such that $f(\vec{a}, b) = M(h, E)$. We put $F(\vec{a}, b, d) = \{b\} \times E$. □

For the previous theorem we need to reprove, for $\mathrm{CRSF}^+_\subseteq$ functions, some technical results from [4] which capture the idea that #-terms behave like monotone functions with respect to embeddings.

Lemma 9.7 *Let f, g, h, i be $\mathrm{CRSF}^+_\subseteq$ functions with arguments $\vec{a}, b$, which we treat as parameters. Let τ_1, τ_2 be $\mathrm{CRSF}^+_\subseteq$ functions with arguments $x, \vec{a}, b$.*

1. *There is a $\mathrm{CRSF}^+_\subseteq$ function τ such that if $\tau_1 : f \preccurlyeq g$ and $\tau_2 : g \preccurlyeq h$, then $\tau : f \preccurlyeq h$.*
2. *There are $\mathrm{CRSF}^+_\subseteq$ functions τ and τ' such that if $\tau_1 : f \preccurlyeq g$ and $\tau_2 : h \preccurlyeq i$, then $\tau : f \odot h \preccurlyeq g \odot i$ and $\tau' : f\#h \preccurlyeq g\#i$.*
3. *If g is a #-term then there is a $\mathrm{CRSF}^+_\subseteq$ function τ with arguments $x, \vec{a}, b, c$ such that if $\tau_1(x, \vec{a}, b) : f(\vec{a}, b) \preccurlyeq g(\vec{a}, b)$ and $b \leq c$ then $\tau(x, \vec{a}, b, c) : f(\vec{a}, b) \preccurlyeq g(\vec{a}, c)$.*

Proof For item 1, let $\tau(x) = \{z \in [h] : \exists y{\in}\tau_1(x),\, z \in \tau_2(y)\}$. The proof that this is an embedding function is just as in Lemma 18 of [4].

For item 2, extend τ_1 and τ_2 so that $\tau_1(f) = \{g\}$ and $\tau_2(h) = \{i\}$. Let

$$\tau(x) = \begin{cases} \tau_2(x) & \text{if } x \in \mathrm{tc}(h) \\ \{y \odot i : y \in \tau_1(x \odot^{-1} h)\} & \text{otherwise} \end{cases}$$

$$\tau'(x) = \{\sigma_{g,i}(y', z') : y' \in \tau_1(\pi_{1,f,h}(x)) \wedge z' \in \tau_2(\pi_{2,f,h}(x))\},$$

where $x \odot^{-1} h$ is the (unique) z such that $x = z \odot h$, or is 0 if no such z exists. Then $\tau(x) \subseteq \mathrm{tc}(g \odot i)$ and $\tau'(x) \subseteq \mathrm{tc}(g\#i)$ so both functions can be computed in $\mathrm{CRSF}^+_\subseteq$ using separation. The proofs that these work are as in Lemma 19 of [4].

For item 3 it is enough, using item 1, to find a function τ' such that $\tau'(x, \vec{a}, b, c) : g(\vec{a}, b) \preccurlyeq g(\vec{a}, c)$ whenever $b \leq c$. This is done by induction on the complexity of the #-term g, using items 1 and 2. □

10 Rudimentary functions

The class Rud of rudimentary functions was introduced in [9], as the smallest class which contains projections, pairing and set subtraction $x \setminus y$ and is closed under composition and the union operation $f(\vec{a}, c) := \bigcup_{b \in c} g(\vec{a}, b)$. We will use some properties of Rud shown in [9], namely that it contains $\times$, is closed under separation, and that the Rud relations are closed under Boolean operations and Δ_0 quantification.

Lemma 10.1 *Every* Rud *function is in* $\mathrm{CRSF}^+_\subseteq$.

Proof The only possible issue is closure under union, but this is taken care of by Corollary 8.5. □

Definition 10.2 *Let* RS *be the class of functions obtained from* Rud *by adding transitive closure as a basic function, and closing under subset-bounded recursion.*

We will show that RS and $\mathrm{CRSF}^+_\subseteq$ have essentially the same complexity. A straightforward induction shows that no function in RS increases the rank of its arguments by more than a constant, so RS does not contain $\odot$ or $\#$ and the two classes cannot exactly coincide. But by Theorem 10.8 below, for any $\mathrm{CRSF}^+_\subseteq$ function f we can compute in RS a string coding the value of f via a version of embedded Mostowski collapse. In particular it follows that the relations in RS and $\mathrm{CRSF}^+_\subseteq$ are the same.

The key idea is that constructions in $\mathrm{CRSF}^+_\subseteq$ typically use recursion over smash-terms ordered by $<$, and we can simulate this in RS using recursion over cartesian products ordered lexicographically by $<$.

Definition 10.3 *For $k \in \mathbb{N}$ and k-tuples $\vec{a}, \vec{b}$ we write $\langle \vec{a} \rangle <_k \langle \vec{b} \rangle$ for the usual lexicographic ordering induced by the ordering $<$ on components. For any set u we write $<^u_k \langle \vec{a} \rangle$ for the set $\{\langle \vec{b} \rangle \in \mathrm{tc}(u) : \langle \vec{b} \rangle <_k \langle \vec{a} \rangle\}$, where $\langle \vec{b} \rangle$ ranges over k-tuples. We write $\langle \vec{b} \rangle <^u_k \langle \vec{a} \rangle$ instead of $\langle \vec{b} \rangle \in <^u_k \langle \vec{a} \rangle$.*

When doing course-of-values recursion we use the notation

$$f_{<^u_k \langle \vec{b} \rangle}(\vec{p}, -) := \bigcup_{\langle \vec{c} \rangle <^u_k \langle \vec{b} \rangle} \{\langle \vec{c} \rangle\} \times f(\vec{p}, \vec{c}).$$

where the dash $-$ indicates the last k arguments of f.

We introduce the bound u because the collection of all tuples $\langle \vec{b} \rangle$ such that $\langle \vec{b} \rangle <_k \langle \vec{a} \rangle$ is typically not a set. Note that $<_k$ is well-founded on k-tuples, in the sense that for any formula φ in the language of set theory, if $\varphi(\vec{p}, \vec{a})$ holds for some k-tuple $\langle \vec{a} \rangle$ then there is some $<_k$ minimal k-tuple $\langle \vec{b} \rangle$ such that $\varphi(\vec{p}, \vec{b})$ holds.

We will use the following strengthening of course-of-values recursion to simulate $\mathrm{CRSF}^+_\subseteq$ in RS.

Definition 10.4 *For $k \in \mathbb{N}$, the function f is obtained from g by* subset-bounded k-lexicographic recursion *with bound h if*

$$f(\vec{p}, u, \vec{a}) = g(\vec{p}, u, \vec{a}, f_{<^u_k \langle \vec{a} \rangle}(\vec{p}, u, -)) \cap h(\vec{p}, u, \vec{a}).$$

We call u the domain *of the recursion.*

Theorem 10.5 *For each $k \in \mathbb{N}$, the class* RS *is closed under subset-bounded k-lexicographic recursion.*

Proof We use induction on k. The case $k = 1$ is just the usual course-of-values recursion as discussed in Section 8. The proof of Corollary 8.8, that this is available in $\mathrm{CRSF}^+_{\subseteq}$, relies on the subset-bounded recursion and union schemes and exactly the same proof goes through for RS.

Suppose $k > 1$. For clarity we will suppress the parameters $\vec{p}$. We will assume that $g(u, \vec{a}, S) \subseteq h(u, \vec{a})$ always, by replacing g with $g \cap h$ if necessary, and we will write $\vec{a}_2$ and $\vec{b}_2$ instead of $a_2, \ldots, a_k$ and $b_2, \ldots, b_k$. We have

$$f(u, \vec{a}) = g(u, \vec{a}, f_{<^u_k \langle \vec{a} \rangle}(u, -))$$

with $g, h \in \mathrm{RS}$. Let f' be defined by the $(k-1)$-lexicographic recursion

$$f'(u, \vec{a}, S) := g(u, \vec{a}, S \cup R(u, a_1, f'_{<^u_{k-1} \langle \vec{a}_2 \rangle}(u, a_1, -, S)))$$

where R is the function

$$R(u, a_1, Z) := \{\langle \langle a_1, \vec{b}_2 \rangle, y \rangle : \langle \langle \vec{b}_2 \rangle, y \rangle \in Z \text{ and } \langle a_1, \vec{b}_2 \rangle \in \mathrm{tc}(u)\}$$

whose purpose is to prepend elements of a course-of-values Z over $<^u_{k-1} \langle \vec{a}_2 \rangle$ with a_1, and remove any which could not then be elements of a course-of-values over $<^u_k \langle \vec{a} \rangle$. By the inductive hypothesis $f' \in \mathrm{RS}$. Define a function F by

$$F(u, a_1) = f_{<^u_k \langle a_1, \vec{0} \rangle}(u, -)$$

where $\vec{0}$ stands for the $(k-1)$-tuple of empty sets.

We claim that for all $\vec{a}$ and u,

$$f'(u, \vec{a}, F(u, a_1)) = f(u, \vec{a}).$$

To see this, fix u and a_1 and use $<_{k-1}$-induction on $\vec{a}_2$. For the inductive step, suppose that for all $\langle \vec{b}_2 \rangle <_{k-1} \langle \vec{a}_2 \rangle$ we have

$$f'(u, a_1, \vec{b}_2, F(u, a_1)) = f(u, a_1, \vec{b}_2).$$

By definition

$$f'(u, a_1, \vec{a}_2, F(u, a_1)) = g(u, a_1, \vec{a}_2, Z) \tag{2}$$

where

$$\begin{aligned}
Z &= F(u,a_1) \ \cup \ R(u,a_1,f'_{<^u_{k-1}\langle \vec{a}_2\rangle}(u,a_1,-,F(u,a_1))) \\
&= f_{<^u_k\langle a_1,\vec{0}\rangle}(u,-) \cup R(u,a_1, \bigcup_{\langle \vec{b}_2\rangle <^u_{k-1}\langle \vec{a}_2\rangle} \{\langle \vec{b}_2\rangle\} \times f'(u,a_1,\vec{b}_2,F(u,a_1)) \\
&= f_{<^u_k\langle a_1,\vec{0}\rangle}(u,-) \ \cup \bigcup_{\substack{\langle \vec{b}_2\rangle <^u_{k-1}\langle \vec{a}_2\rangle \\ \langle a_1,\vec{b}_2\rangle \in \mathrm{tc}(u)}} \{\langle a_1,\vec{b}_2\rangle\} \times f'(u,a_1,\vec{b}_2,F(u,a_1)) \\
&= f_{<^u_k\langle a_1,\vec{0}\rangle}(u,-) \ \cup \bigcup_{\substack{\langle \vec{b}_2\rangle <^u_{k-1}\langle \vec{a}_2\rangle \\ \langle a_1,\vec{b}_2\rangle \in \mathrm{tc}(u)}} \{\langle a_1,\vec{b}_2\rangle\} \times f(u,a_1,\vec{b}_2)
\end{aligned}$$

by the inductive hypothesis. From the definition,

$$\begin{aligned}
<^u_k \langle \vec{a}\rangle &= \{\langle b_1,\vec{b}_2\rangle \in \mathrm{tc}(u) : (b_1 < a_1) \vee (b_1 = a_1 \wedge \langle \vec{b}_2\rangle <_{k-1} \langle \vec{a}_2\rangle)\} \\
&= <^u_k \langle a_1,\vec{0}\rangle \cup \{\langle a_1,\vec{b}_2\rangle \in \mathrm{tc}(u) : \langle \vec{b}_2\rangle <^u_{k-1} \langle \vec{a}_2\rangle\}
\end{aligned}$$

where we are using that $\langle a_1,\vec{b}_2\rangle \in \mathrm{tc}(u)$ implies $\langle \vec{b}_2\rangle \in \mathrm{tc}(u)$. It follows that $Z = f_{<^u_k\langle \vec{a}\rangle}(u,-)$. Therefore the right hand side of (2) is exactly the recursive definition of $f(u,\vec{a})$, giving us the inductive step.

To finish the proof it is enough to show that F is in RS. We have

$$F(u,a_1) = f_{<^u_k\langle a_1,\vec{0}\rangle}(u,-) = \bigcup_{\substack{\langle \vec{b}\rangle \in \mathrm{tc}(u) \\ b_1 < a_1}} \{\langle \vec{b}\rangle\} \times f(u,\vec{b}) = \bigcup_{\substack{\langle \vec{b}\rangle \in \mathrm{tc}(u) \\ b_1 < a_1}} \{\langle \vec{b}\rangle\} \times f'(u,\vec{b},F(u,b_1))$$

hence F is definable by course-of-values recursion on a_1, using f'. Furthermore since $f \subseteq h$ we have $F(u,a_1) \subseteq \mathrm{tc}(u) \times \bigcup_{\langle \vec{b}\rangle <^u_k \langle a_1,\vec{0}\rangle} h(u,\vec{b})$, and this bound is in RS. Hence F is in RS. □

We now generalize the notion of diagram from Definition 2.1.

Definition 10.6 *For $k \in \mathbb{N}$, a* k-diagram *is a pair $\langle \langle \vec{a}\rangle, E\rangle$ where $\langle \vec{a}\rangle$ is a k-tuple, and a pair $\langle \langle \vec{x}\rangle, \langle \vec{y}\rangle\rangle$ of k-tuples is in E only if $\langle \vec{x}\rangle <_k \langle \vec{y}\rangle$. The* domain *of a k-diagram is defined as*

$$\mathrm{dom}(\langle \langle \vec{a}\rangle, E\rangle) = \{\langle \vec{a}\rangle\} \cup \{\langle \vec{x}\rangle : \textit{there is some pair } \langle \langle \vec{x}\rangle, \langle \vec{y}\rangle\rangle \in E\}.$$

A k-diagram represents the graph with nodes $\mathrm{dom}(\langle \langle \vec{a}\rangle, E\rangle)$ and edges given by E.

Definition 10.7 *For $k \in \mathbb{N}$, the* k-embedded Mostowski collapse M_k *is defined by the lexicographic recursion*

$$M_k(\langle \vec{a} \rangle, E) = \{M_k(\langle \vec{b} \rangle, E) : \langle \langle \vec{b} \rangle, \langle \vec{a} \rangle \rangle \in E \text{ and } \langle \vec{b} \rangle <_k \langle \vec{a} \rangle\}.$$

For convenience we will often treat M_k as a one-argument function, writing $M_k(\Delta)$ to mean $M_k(\langle \vec{a} \rangle, E)$ for a k-diagram $\Delta = \langle \langle \vec{a} \rangle, E \rangle$. We consider a k-diagram Δ as a code for the set $M_k(\Delta)$. Working with codes rather than directly with sets allows us to simulate computations involving sets of higher rank than we could produce in RS. Note that M_k itself is not in RS, at least for $k \geq 2$, and that if we add M_2 to RS it becomes possible to construct # and thus every function in $\mathrm{CRSF}^+_\subseteq$.

We state our main result.

Theorem 10.8 *If $f \in \mathrm{CRSF}^+_\subseteq$ then there is a function $F \in$ RS and $k \in \mathbb{N}$ such that $F(\vec{a})$ is a k-diagram and $f(\vec{a}) = M_k(F(\vec{a}))$. It follows that the $\mathrm{CRSF}^+_\subseteq$ and* RS *relations are the same, and furthermore that for $f \in \mathrm{CRSF}^+_\subseteq$ the function $g(\vec{a}, b) := f(\vec{a}) \cap b$ is in* RS.

This will follow from Theorem 10.10 below, together with the observation that RS contains the function $a \mapsto \langle a, \in\restriction[a]^2 \rangle$ which maps a to a 1-diagram coding a. For the last sentence we repeat the proof of Corollary 8.3, using Lemma 10.11 to compute bisimulations in RS.

Definition 10.9 *We say that a function $f(x_1, \ldots, x_m)$ is* RS-definable on diagrams *if for all $k_1, \ldots, k_m \in \mathbb{N}$ there is an* RS *function $F_{\vec{k}}(x_1, \ldots, x_m)$ and $\ell \in \mathbb{N}$ such that, for all $\Delta_1, \ldots, \Delta_m$ where each Δ_i is a k_i-diagram, $F_{\vec{k}}(\Delta_1, \ldots, \Delta_m)$ is an ℓ-diagram and*

$$M_\ell(F_{\vec{k}}(\Delta_1, \ldots, \Delta_m)) = f(M_{k_1}(\Delta_1), \ldots, M_{k_m}(\Delta_m)).$$

A relation $r(x_1, \ldots, x_m)$ is RS-definable on diagrams *if for all $k_1, \ldots, k_m \in \mathbb{N}$ there is an* RS *relation $R_{\vec{k}}(x_1, \ldots, x_m)$ such that for all k_i-diagrams Δ_i*

$$R_{\vec{k}}(\Delta_1, \ldots, \Delta_m) \Leftrightarrow r(M_{k_1}(\Delta_1), \ldots, M_{k_m}(\Delta_m)).$$

We show that RS can simulate $\mathrm{CRSF}^+_\subseteq$ functions in this sense.

Theorem 10.10 *Every $\mathrm{CRSF}^+_\subseteq$ function is* RS*-definable on diagrams.*

The proof of this takes up the rest of this section. The constructions used are similar to those in Section 7.

Lemma 10.11 *For $k, \ell \in \mathbb{N}$, there is a function $B_{k,\ell}(\langle\vec{a}\rangle, E, \langle\vec{b}\rangle, F)$ in* RS *which computes the bisimulation between the k-diagram $\langle\langle\vec{a}\rangle, E\rangle$ and the ℓ-diagram $\langle\langle\vec{b}\rangle, F\rangle$.*

Proof First observe that the desired bisimulation is a subset of $\mathrm{dom}(\langle\vec{a}\rangle, E) \times \mathrm{dom}(\langle\vec{b}\rangle, F)$. We will define $B_{k,\ell}$ by k-lexicographic recursion on $\langle\vec{a}\rangle$ with domain E. Suppose we are given $S = (B_{k,\ell})_{\restriction <^E_k \langle\vec{a}\rangle}(-, E, \langle\vec{b}\rangle, F)$, that is, the course-of-values of $B_{k,\ell}$ below $\langle\vec{a}\rangle$. Expanding the definition,

$$S = \bigcup_{\substack{\langle\vec{c}\rangle <_k \langle\vec{a}\rangle \\ \langle\vec{c}\rangle \in \mathrm{tc}(E)}} \{\langle\vec{c}\rangle\} \times B_{k,\ell}(\langle\vec{c}\rangle, E, \langle\vec{b}\rangle, F).$$

Let $\langle\vec{c}\rangle$ be any E-predecessor of $\langle\vec{a}\rangle$. Then $\langle\vec{c}\rangle <_k \langle\vec{a}\rangle$ and $\langle\vec{c}\rangle \in \mathrm{tc}(E)$. Hence we can extract from S the bisimulation between $\langle\langle\vec{c}\rangle, E\rangle$ and $\langle\langle\vec{b}\rangle, F\rangle$ using an RS function, as

$$B_{k,\ell}(\langle\vec{c}\rangle, E, \langle\vec{b}\rangle, F) = \{z \in \bigcup\bigcup S : \langle\langle\vec{c}\rangle, z\rangle \in S\}.$$

We then carry on as in the proof of Lemma 5.7. □

Corollary 10.12 *The relations $=$ and $\in$ are* RS*-definable on diagrams.*

Lemma 10.13 *The constant* 0 *and the functions pairing,* $\mathrm{cond}_\in$ *$\times$, $\odot$ and $\#$ are* RS*-definable on diagrams.*

Proof A 1-diagram for 0 is simply $\langle 0, 0\rangle$. For the other functions, suppose we are given a k-diagram $\Gamma = \langle\langle\vec{a}\rangle, E\rangle$ and an ℓ-diagram $\Delta = \langle\langle\vec{b}\rangle, F\rangle$.

For the pairing function we take $m = 1 + k + \ell$ and construct an m-diagram, whose nodes we will write in the form $\langle i, \vec{x}, \vec{y}\rangle$ where i is a 1-tuple, $\vec{x}$ is a k-tuple and $\vec{y}$ is an ℓ-tuple. We define an edge relation G, using infix notation for G, by

1. $\langle 0, \vec{x}, \vec{0}\rangle\ G\ \langle 0, \vec{x}', \vec{0}\rangle$ if and only if $\langle\vec{x}\rangle\ E\ \langle\vec{x}'\rangle$
2. $\langle 1, \vec{0}, \vec{y}\rangle\ G\ \langle 1, \vec{0}, \vec{y}'\rangle$ if and only if $\langle\vec{y}\rangle\ F\ \langle\vec{y}'\rangle$
3. $\langle 0, \vec{a}, \vec{0}\rangle\ G\ \langle 2, \vec{0}, \vec{0}\rangle$
4. $\langle 1, \vec{0}, \vec{b}\rangle\ G\ \langle 2, \vec{0}, \vec{0}\rangle$.

Note that G respects the ordering $<_m$. By 1 and 2, $M_m(\langle 0, \vec{a}, \vec{0}\rangle, G) = M_k(\Gamma)$ and $M_m(\langle 1, \vec{0}, \vec{b}\rangle, G) = M_k(\Delta)$. Thus by 3 and 4, $M_m(\langle 2, \vec{0}, \vec{0}\rangle, G) = \{M_k(\Gamma), M_\ell(\Delta)\}$. The set composition and $\mathrm{cond}_\in$ functions are similar (see Lemmas 7.5 and 7.6).

For the smash function $\#$ we take $m = k + \ell$. Using similar notation for tuples as the previous case, we define an edge relation G on m-tuples by

1. $\langle \vec{x}, \vec{y} \rangle \; G \; \langle \vec{x}, \vec{y}' \rangle$ if and only if $\langle \vec{x} \rangle \in \mathrm{dom}(\Gamma)$ and $\langle \vec{y} \rangle \; F \; \langle \vec{y}' \rangle$
2. $\langle \vec{x}, \vec{b} \rangle \; G \; \langle \vec{x}', \vec{y} \rangle$ if and only if $\langle \vec{x} \rangle \; E \; \langle \vec{x}' \rangle$ and $\langle \vec{y} \rangle$ is a source node of F.

We output $\langle \langle \vec{a}, \vec{b} \rangle, G \rangle$.

For the cartesian product we take $m = 3 + k + \ell$. We define an edge relation G by first, for each pair $p \in \mathrm{dom}(\Gamma) \times \mathrm{dom}(\Delta)$, putting

1. $\langle 0, p, 0, \vec{x}, \vec{0} \rangle \; G \; \langle 0, p, 0, \vec{x}', \vec{0} \rangle$ if and only if $\langle \vec{x} \rangle \; E \; \langle \vec{x}' \rangle$
2. $\langle 0, p, 1, \vec{0}, \vec{y} \rangle \; G \; \langle 0, p, 1, \vec{0}, \vec{y}' \rangle$ if and only if $\langle \vec{y} \rangle \; F \; \langle \vec{y}' \rangle$.

For each such p, if p has the form $\langle \langle \vec{x} \rangle, \langle \vec{y} \rangle \rangle$ where $\langle \vec{x} \rangle \; E \; \langle \vec{a} \rangle$ and $\langle \vec{y} \rangle \; F \; \langle \vec{b} \rangle$, meaning that $e_{\langle \vec{x} \rangle} := M_k(\langle \vec{x} \rangle, E) \in M_k(\Gamma)$ and $f_{\langle \vec{y} \rangle} := M_\ell(\langle \vec{y} \rangle, F) \in M_\ell(\Delta)$, then we add six more edges:

3. $\langle 0, p, 0, \vec{x}, \vec{0} \rangle \; G \; \langle 0, p, 2, \vec{0}, \vec{0} \rangle$
4. $\langle 0, p, 0, \vec{x}, \vec{0} \rangle \; G \; \langle 0, p, 3, \vec{0}, \vec{0} \rangle$ and $\langle 0, p, 1, \vec{0}, \vec{y} \rangle \; G \; \langle 0, p, 3, \vec{0}, \vec{0} \rangle$
5. $\langle 0, p, 2, \vec{0}, \vec{0} \rangle \; G \; \langle 0, p, 4, \vec{0}, \vec{0} \rangle$ and $\langle 0, p, 3, \vec{0}, \vec{0} \rangle \; G \; \langle 0, p, 4, \vec{0}, \vec{0} \rangle$
6. $\langle 0, p, 4, \vec{0}, \vec{0} \rangle \; G \; \langle 1, 0, 0, \vec{0}, \vec{0} \rangle$.

From 1 and 3, we have $M_m(\langle 0, p, 2, \vec{0}, \vec{0} \rangle, G) = \{e_{\langle \vec{x} \rangle}\}$. From 1, 2 and 4, we have $M_m(\langle 0, p, 3, \vec{0}, \vec{0} \rangle, G) = \{e_{\langle \vec{x} \rangle}, f_{\langle \vec{y} \rangle}\}$. From 5, $M_m(\langle 0, p, 4, \vec{0}, \vec{0} \rangle, G) = \langle e_{\langle \vec{x} \rangle}, f_{\langle \vec{y} \rangle} \rangle$. We output $\langle \langle 1, 0, 0, \vec{0}, \vec{0} \rangle, G \rangle$, which thus by 6 codes the whole set $M_k(\Gamma) \times M_\ell(\Delta)$. □

Below we will use small Greek letters to denote tuples of sets. The arity will be clear from the context.

Lemma 10.14 *The transitive closure function is* RS*-definable on diagrams.*

Proof Given a k-diagram $\Delta = \langle \alpha, E \rangle$, we first claim that there is an RS function computing the ancestor relation on the nodes of Δ.

To see this, for $\beta \leq_k \alpha$ let R_β be the ancestor relation on the graph whose nodes are the set $\leq_k^{\mathrm{dom}(\Delta)} \beta$ and whose edges are those induced on this set by E. Then $R_\beta \subseteq \mathrm{dom}(\Delta) \times \mathrm{dom}(\Delta)$ and can be computed by subset-bounded k-lexicographic recursion, where the form of the recursion is similar to the proof of Lemma 10.11, and we compute R_β from earlier values R_γ with $\langle \gamma, \beta \rangle \in E$ as in Lemma 6.15.

We output $\langle \alpha, E \cup G \rangle$ where G consists of every pair $\langle \beta, \alpha \rangle$ such that β is an E-ancestor of α. □

Lemma 10.15 *Embedded Mostowski collapse is* RS*-definable on diagrams.*

Proof We are given a k-diagram $\langle \alpha, G_a \rangle$ and an ℓ-diagram $\langle \beta, G_E \rangle$. We will construct a set H such that

$$M_k(\langle \alpha, H \rangle) = M(a, E)$$

where $a = M_k(\langle \alpha, G_a \rangle)$ and $E = M_\ell(\langle \beta, G_E \rangle)$. As in Lemma 7.10, we define $\kappa(\rho) = M_k(\langle \rho, G_a \rangle)$ for k-tuples ρ. We put

$$H = \{\langle \rho, \sigma \rangle \in \mathrm{dom}(\langle \alpha, G_a \rangle) \times \mathrm{dom}(\langle \alpha, G_a \rangle) : \\ \rho <_k \sigma \wedge \kappa(\rho) < \kappa(\sigma) \wedge \langle \kappa(\rho), \kappa(\sigma) \rangle \in E\}.$$

The proof that this works is the same as for Lemma 7.10.

To construct H in RS, we first observe that $\kappa(\rho) < \kappa(\sigma)$ if and only if $M_k(\langle \rho, G_a \rangle) \in \mathrm{tc}(M_k(\langle \sigma, G_a \rangle))$. Since membership and transitive closure are RS-definable on diagrams, this relation is as well. To decide whether $\langle \kappa(\rho), \kappa(\sigma) \rangle \in E$, we first construct a subcircuit computing the bisimulation $\sim$ between $\langle \alpha, G_a \rangle$ and $\langle \beta, G_E \rangle$. Then $\langle M_k(\langle \rho, G_a \rangle), M_k(\langle \sigma, G_a \rangle) \rangle$ is in E if and only if there exist ℓ-tuples ρ', σ' in $\mathrm{dom}(\langle \beta, G_E \rangle)$ such that

$$\rho \sim \rho' \ \wedge \sigma \sim \sigma' \wedge \langle M_\ell(\langle \rho', G_E \rangle), M_\ell(\langle \sigma', G_E \rangle) \rangle \in M_\ell(\langle \beta, G_E \rangle)$$

and this condition is decidable in RS, since membership and the pairing function relation are RS-definable on diagrams. □

Lemma 10.16 *For $k \in \mathbb{N}$ there is an* RS *function* $\mathrm{Collect}_k$ *such that, for any set S of k-diagrams,* $\mathrm{Collect}_k(S)$ *is a $(k+1)$-diagram with*

$$M_{k+1}(\mathrm{Collect}_k(S)) = \{M_k(\Delta) : \Delta \in S\}.$$

Proof We write a k-diagram Δ as $\langle \langle \vec{a}_\Delta \rangle, E_\Delta \rangle$, and can recover the k-tuple $\vec{a}_\Delta$ and the set E_Δ from Δ using RS functions. Define

$$F = \bigcup_{\Delta \in S} \{\langle \langle \Delta, \vec{x} \rangle, \langle \Delta, \vec{x}' \rangle \rangle : \langle \langle \vec{x} \rangle, \langle \vec{x}' \rangle \rangle \in E_\Delta\}.$$

That is, for each pair in E_Δ we prepend Δ to both k-tuples in the pair, turning them into $(k+1)$-tuples, and then take the union over all $\Delta \in S$. The result is that $M_{k+1}(\langle \langle \Delta, \vec{a}_\Delta \rangle, F \rangle) = M_k(\Delta)$ for each $\Delta \in S$. Then we define

$$G = F \cup \{\langle \langle \Delta, \vec{a}_\Delta \rangle, \langle S, \vec{0} \rangle \rangle : \Delta \in S\}$$

where $\vec{0}$ is a k-tuple of empty sets. Note that the extra edges respect the lexicographic ordering. We output $\langle \langle S, \vec{0} \rangle, G \rangle$. □

Lemma 10.17 *Suppose g and h are* RS*-definable on diagrams. Then the function f defined by subset-bounded recursion from g with bound h is* RS*-definable on diagrams.*

Proof For simplicity of presentation we will only consider a recursion without parameters — this changes nothing important. So we have

$$f(x) = g(\{f(y) : y \in x\}, x) \cap h(x).$$

Suppose that our input x is given as a k-diagram. By assumption there are $\ell, m \in \mathbb{N}$ and a function $H \in \mathrm{RS}$ simulating h, which takes as input a k-diagram and outputs an ℓ-diagram, and a function $G \in \mathrm{RS}$ simulating g, which takes as input an $(\ell{+}1)$-diagram (for the previous values) and a k-diagram (for x) and outputs an m-diagram.

Let us name the parts of a diagram, so that $\Delta = \langle \mathrm{sink}(\Delta), \mathrm{Edges}(\Delta)\rangle$. We claim that there is an RS function G' such that for all k-diagrams Δ and $(\ell{+}1)$-diagrams Γ, we have that $G'(\Gamma, \Delta) \subseteq \mathrm{Edges}(H(\Delta))$ and

$$M_\ell(\langle \mathrm{sink}(H(\Delta)), G'(\Gamma, \Delta)\rangle) = g(M_{\ell+1}(\Gamma), M_k(\Delta)) \cap h(M_k(\Delta)). \quad (3)$$

To compute G', we first compute $G(\Gamma, \Delta)$, $H(\Delta)$ and the bisimulation $\sim$ between them. To deal with the intersection, we then take the set $\mathrm{Edges}(H(\Delta))$ and delete from it every edge $\langle\langle \vec{v}\rangle, \mathrm{sink}(H(\Delta))\rangle$ for which there is no $\langle \vec{w}\rangle \sim \langle \vec{v}\rangle$ with the edge $\langle\langle \vec{w}\rangle, \mathrm{sink}(G(\Gamma, \Delta))\rangle$ in $\mathrm{Edges}(G(\Gamma, \Delta))$. We output the result.

We can now define a function simulating f by recursively applying G'. Below, α, β, γ stand for k-tuples. Suppose our input is $\langle \alpha, E\rangle$. Define a function $F(\beta, E)$, by k-lexicographic recursion on β with domain E as

$$F(\beta, E) = G'(\mathrm{Collect}_\ell(S_\beta), \langle \beta, E\rangle)$$
$$\text{where } S_\beta = \{\langle s(\gamma), F(\gamma, E)\rangle : \gamma <_k^E \beta, \langle \gamma, \beta\rangle \in E\}$$

and we introduce the notation $s(\gamma)$ for $\mathrm{sink}(H(\langle \gamma, E\rangle))$. Recall that the course-of-values of $F(-, E)$ below β on domain E is

$$F_{<_k^E \beta}(-, E) := \bigcup_{\gamma <_k^E \beta} \{\gamma\} \times F(\gamma, E),$$

from which we can construct S_β in RS. Furthermore this is a subset-bounded recursion, since by the definition of G' we have that $F(\beta, E) \subseteq \mathrm{Edges}(H(\langle \beta, E\rangle))$. Hence F is in RS.

We claim that $M_\ell(\langle s(\beta), F(\beta, E)\rangle) = f(M_k(\langle \beta, E\rangle))$. It follows that we can define f on diagrams by setting β to be α. We will use k-lexicographic induction on β. Letting $\Delta = \langle \beta, E\rangle$ and $b = M_k(\Delta)$, we have

$$\begin{aligned}
M_\ell(\langle s(\beta), F(\beta, E)\rangle) &= M_\ell(\langle \mathrm{sink}(H(\Delta)), G'(\mathrm{Collect}_\ell(S_\beta), \Delta)\rangle) \\
&= g(M_{l+1}(\mathrm{Collect}_\ell(S_\beta)), b) \cap h(b) \\
&= g(\{M_\ell(\langle s(\gamma), F(\gamma, E)\rangle) : \gamma <_k^E \beta, \langle \gamma, \beta\rangle \in E\}, b) \cap h(b) \\
&= g(\{f(M_k(\langle \gamma, E\rangle)) : \gamma <_k^E \beta, \langle \gamma, \beta\rangle \in E\}, b) \cap h(b) \\
&= g(\{f(c) : c \in b\}, b) \cap h(b) \\
&= f(b)
\end{aligned}$$

using, in order, the definition of F, equation (3), the definition of $\mathrm{Collect}_\ell$, the inductive hypothesis, and the recursive definitions of M_k and f. □

References

[1] P. Aczel, *Non-Well-Founded Sets*, CSLI Lecture Notes, no. 14, Center for the Study of Language and Information, Stanford, 1988.

[2] T. Arai, *Predicatively computable functions on sets*, Archive for Mathematical Logic, 54 (2015), pp. 471–485.

[3] A. Beckmann, S. R. Buss, and S.-D. Friedman, *Safe recursive set functions*, Journal of Symbolic Logic, 80 (2015), pp. 730–762.

[4] A. Beckmann, S. R. Buss, S.-D. Friedman, M. Müller, and N. Thapen, *Cobham recursive set functions*, Annals of Pure and Applied Logic, 167 (2016), pp. 335–369.

[5] ——, *Cobham recursive set functions and weak set theories*, in Sets and Computations, Lecture Notes Series, Institute for Mathematical Sciences, National University of Singapore, World Scientific, 2017, pp. 55–116.

[6] A. Cobham, *The intrinsic computational difficulty of functions*, in Logic, Methodology and Philosophy of Science, Proceedings of the Second International Congress, held in Jerusalem, 1964, Y. Bar-Hillel, ed., Amsterdam, 1965, North-Holland, pp. 24–30.

[7] J. D. Hamkins and A. Lewis, *Infinite time Turing machines*, Journal of Symbolic Logic, 65 (2000), pp. 567–604.

[8] J. Håstad, *Almost optimal lower bounds for small depth circuits*, in Proceedings of the 18-th Annual ACM Symposium on Theory of Computing, 1986, pp. 6–20.

[9] R. B. Jensen, *The fine structure of the constructible hierarchy*, Annals of Mathematical Logic, 4 (1972), pp. 229–308. Errata, ibid **4** (1972) 443.

[10] G. E. Sacks, *Higher recursion theory*, Springer-Verlag, Berlin, 1990.

[11] V. Y. Sazonov, *On bounded set theory*, in Logic and Scientific Methods, M. L. D. C. et al., ed., Synthese Library Volume 259, Kluwer Academic, 1997, pp. 85–103.

[12] R. Schindler, *$P \neq NP$ for infinite time Turing machines*, Monatshefte für Mathematik, 139 (2003), pp. 335–340.