Small circuits for feasible set functions

Neil Thapen

Institute of Mathematics
Czech Academy of Sciences

Joint work with

Arnold Beckmann, Sam Buss, Sy Friedman and Moritz Müller

Some previous work

- Sazonov – Bounded set theory
- Hamkins and Lewis – Infinite time Turing machines
- Beckmann, Buss, Friedman – Safe recursive set functions (SRSF)
- Arai – Predicatively computable set functions (PCSF)

Introduction - CRSF

The Cobham recursive set functions (CRSF) are defined by taking some basic initial functions and closing under composition and a limited recursion.

Introduction - CRSF

The Cobham recursive set functions (CRSF) are defined by taking some basic initial functions and closing under composition and a limited recursion.

We can only use recursion to introduce a new function if we know that the new function is "no more complex" than a function already in CRSF.

Introduction - CRSF

The Cobham recursive set functions (CRSF) are defined by taking some basic initial functions and closing under composition and a limited recursion.

We can only use recursion to introduce a new function if we know that the new function is "no more complex" than a function already in CRSF.

We add a smash function $\#$ as an initial function, to allow "polynomial increase in complexity".

Introduction - CRSF

The Cobham recursive set functions (CRSF) are defined by taking some basic initial functions and closing under composition and a limited recursion.

We can only use recursion to introduce a new function if we know that the new function is "no more complex" than a function already in CRSF.

We add a smash function $\#$ as an initial function, to allow "polynomial increase in complexity".

There is rather little set theory involved. CRSF also make sense as a way of talking about parallel computing where the inputs and outputs are directed acyclic graphs.

Introduction - results

Arai recently introduced a class PCSF of set functions which capture polynomial time if restricted to finite binary strings.

Theorem

CRSF $=$ PCSF$^+$, which is a slightly relaxed version of PCSF.

Introduction - results

Theorem

CRSF contains exactly the functions from sets to sets which are computed by uniform “polynomial size” families of infinite circuits.

Introduction - results

We define a weak fragment of Kripe-Platek set theory $\mathsf{KP}_1^{\preccurlyeq}$, modelled on the bounded arithmetic theory S_2^1.

It weakens Foundation to $\in$-induction for formulas of the form

$$\exists y \preccurlyeq t(x)\, \theta(x, y)$$

where $\theta \in \Delta_0$.

Almost-Theorem

CRSF contains exactly the Σ_1-definable functions of $\mathsf{KP}_1^{\preccurlyeq}$.

Introduction - results

Define a function class RS by taking Jensen's rudimentary set functions Rud, adding transitive closure, and closing under subset-bounded recursion.

(We do not add the smash function $\#$)

Theorem

The functions in RS and CRSF are the same, modulo a certain way of coding the output.

In particular, RS and CRSF contain the same $0/1$-valued functions.

Outline

Outline

- The definition of CRSF

Outline

- The definition of CRSF
- Turing machines

Outline

- The definition of CRSF
- Turing machines
- Circuits

Cobham's definition of P

The smallest set of functions from binary strings to binary strings which contains as initial functions

- constant ϵ (empty string)
- functions $s \mapsto s0$ and $s \mapsto s1$
- projection functions $a_1, \ldots, a_n \mapsto a_i$
- the smash function $a, b \mapsto a\#b = 0^{|a||b|}$

and is closed under composition and recursion, where we introduce a new function f from functions g, h_0, h_1, k in P by

$$f(\bar{a}, \epsilon) = g(\bar{a})$$
$$f(\bar{a}, s0) = h_0(\bar{a}, f(s), s)$$
$$f(\bar{a}, s1) = h_1(\bar{a}, f(s), s)$$

provided that $|f(\bar{a}, s)| \leq k(|a_1|, \ldots, |a_n|, |s|)$ for all $\bar{a}, s$.

The primitive recursive set functions

Initial functions are:

- projections: $a_1, \ldots, a_n \mapsto a_i$
- conditional: $\mathrm{cond}_\in(a, b, c, d) = a$ if $c \in d$, or b otherwise
- pair: $a, b \mapsto \{a, b\}$
- empty set: $\emptyset$
- union: $a \mapsto \bigcup a$

These are closed under composition and recursion:

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

The primitive recursive set functions

Initial functions are:

- projections: $a_1, \ldots, a_n \mapsto a_i$
- conditional: $\mathrm{cond}_\in(a, b, c, d) = a$ if $c \in d$, or b otherwise
- pair: $a, b \mapsto \{a, b\}$
- empty set: $\emptyset$
- union: $a \mapsto \bigcup a$

and we may add:

- transitive closure: $a \mapsto \mathrm{tc}(a)$
- cartesian product: $a, b \mapsto a \times b$

These are closed under composition and recursion:

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

The Cobham recursive set functions

Initial functions are:

. . . as before, plus

- set composition: $a, b \mapsto a \odot b$
- set smash: $a, b \mapsto a \# b$
- embedded Mostowski collapse: $a, E \mapsto M(a, E)$

These are closed under composition and *subset-bounded* recursion.

This is defined by: if $g, h \in$ CRSF then so is

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

Mostowski graphs

The *Mostowski graph* of a set a is the directed graph $\langle \mathrm{tc}(\{a\}), E\rangle$ with nodes $\mathrm{tc}(\{a\})$ and edges E given by $\langle x, y\rangle \in E \longleftrightarrow x \in y$.

We denote it by $\mathcal{G}(a)$. It has a unique source 0 and sink a.

Examples: $\mathcal{G}(2)$, $\mathcal{G}(\{1, 2\})$, $\mathcal{G}(\omega)$

Extra initial functions – composition $\odot$

Definition

Given sets a, b the set composition $a \odot b$ is the set whose Mostowski graph is constructed as follows:

- Draw the graph $\mathcal{G}(a)$ above the graph $\mathcal{G}(b)$
- Identify the source of $\mathcal{G}(a)$ with the sink of $\mathcal{G}(b)$.

Extra initial functions – composition $\odot$

Definition

Given sets a, b the set composition $a \odot b$ is the set whose Mostowski graph is constructed as follows:

- Draw the graph $\mathcal{G}(a)$ above the graph $\mathcal{G}(b)$
- Identify the source of $\mathcal{G}(a)$ with the sink of $\mathcal{G}(b)$.

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

Extra initial functions – composition $\odot$

Definition

Given sets a, b the set composition $a \odot b$ is the set whose Mostowski graph is constructed as follows:

- Draw the graph $\mathcal{G}(a)$ above the graph $\mathcal{G}(b)$
- Identify the source of $\mathcal{G}(a)$ with the sink of $\mathcal{G}(b)$.

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

Notice

1. $\mathrm{rank}(a \odot b) = \mathrm{rank}(b) + \mathrm{rank}(a)$
2. $|\mathrm{tc}(a \odot b)| = |\mathrm{tc}(a)| + |\mathrm{tc}(b)|$.

Extra initial functions – smash $\#$

Definition

Given sets a, b the set smash $a\#b$ is the set whose Mostowski graph is constructed as follows:

- Draw a disjoint copy G_x of $\mathcal{G}(b)$ for every node $x \in \mathcal{G}(a)$
- For each edge $\langle x, y\rangle$ of $\mathcal{G}(a)$, connect the sink of G_x to the source of G_y.

Extra initial functions – smash $\#$

Definition

Given sets a, b the set smash $a\#b$ is the set whose Mostowski graph is constructed as follows:

- Draw a disjoint copy G_x of $\mathcal{G}(b)$ for every node $x \in \mathcal{G}(a)$
- For each edge $\langle x, y \rangle$ of $\mathcal{G}(a)$, connect the sink of G_x to the source of G_y.

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

Extra initial functions – smash

Definition

Given sets a, b the set smash $a\#b$ is the set whose Mostowski graph is constructed as follows:

- Draw a disjoint copy G_x of $\mathcal{G}(b)$ for every node $x \in \mathcal{G}(a)$
- For each edge $\langle x, y\rangle$ of $\mathcal{G}(a)$, connect the sink of G_x to the source of G_y.

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

Notice

1. $\mathrm{rank}(a\#b) + 1 = (\mathrm{rank}(b) + 1)(\mathrm{rank}(a) + 1)$
2. $|\mathrm{tc}(\{a\#b\})| = |\mathrm{tc}(\{a\})| \cdot |\mathrm{tc}(\{b\})|$.

Extra initial functions – embedded Mostowski collapse

. . . we will come back to this

Basic properties of CRSF

Definition

The CRSF relations are relations of the form $f(\bar{a}) \neq 0$ for a CRSF function f.

Lemma

1. CRSF is closed under separation. That is, if $\varphi(\bar{a}, c)$ is a CRSF relation then the following function is in CRSF:

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

2. The CRSF relations are closed under Δ_0 quantification.
3. CRSF contains the rank function, and ordinal addition and multiplication.

Basic properties of CRSF

Definition

The CRSF relations are relations of the form $f(\bar{a}) \neq 0$ for a CRSF function f.

Lemma

1. CRSF is closed under separation. That is, if $\varphi(\bar{a}, c)$ is a CRSF relation then the following function is in CRSF:

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

2. The CRSF relations are closed under Δ_0 quantification.
3. CRSF contains the rank function, and ordinal addition and multiplication.
4. CRSF does not contain ordinal exponentiation.

Turing machines

Turing machines

Let $n, t \in \omega$. Consider a Turing machine limited to tape length n, in the binary language.

A configuration of the machine is a binary string of length n (if we code just the tape contents, and ignore the state and the head position).

Turing machines

Let $n, t \in \omega$. Consider a Turing machine limited to tape length n, in the binary language.

A configuration of the machine is a binary string of length n (if we code just the tape contents, and ignore the state and the head position).

We represent such a binary string as a subset of the ordinal n.

E.g. we represent $01101 \in \{0,1\}^5$ by $\{1,2,4\} \subseteq 5$.

Turing machines

Let $n, t \in \omega$. Consider a Turing machine limited to tape length n, in the binary language.

A configuration of the machine is a binary string of length n (if we code just the tape contents, and ignore the state and the head position).

We represent such a binary string as a subset of the ordinal n. E.g. we represent $01101 \in \{0,1\}^5$ by $\{1,2,4\} \subseteq 5$.

Similarly, we can represent a complete history of a computation up to time t as a subset $W \subseteq (t+1) \times n$.

We can recover the configuration at time i as $\{x \in n : \langle i, x \rangle \in W\}$.

Turing machines

Lemma

Let the function $f : E, n, t \mapsto W$ take as input numbers $n, t \in \omega$ and an initial configuration $E \subseteq n$ of the machine, and output the computation up to time t. Then $f \in CRSF$.

Turing machines

Lemma

Let the function $f : E, n, t \mapsto W$ take as input numbers $n, t \in \omega$ and an initial configuration $E \subseteq n$ of the machine, and output the computation up to time t. Then $f \in CRSF$.

Proof

We use recursion to compute successively

> the computation up to time 0 (this is given by E)
>
> the computation up to time 1
>
> ...
>
> the computation up to time t.

These are all subsets of $(t+1) \times n$, so subset bounded recursion is sufficient.

It is easy to extend by one computation step, using separation with a Δ_0 formula.

Turing machines

Let us continue to identify finite binary strings with subsets of finite ordinals.

Turing machines

Let us continue to identify finite binary strings with subsets of finite ordinals.

Theorem

If we restrict the input and output to subsets of finite ordinals, then CRSF is exactly polynomial time.

Turing machines

Let us continue to identify finite binary strings with subsets of finite ordinals.

Theorem

If we restrict the input and output to subsets of finite ordinals, then CRSF is exactly polynomial time.

Proof

$\mathbf{P} \subseteq \mathbf{CRSF}$: We use the lemma, plus ordinal arithmetic to compute the bounds on time and tape length.

Turing machines

Let us continue to identify finite binary strings with subsets of finite ordinals.

Theorem

If we restrict the input and output to subsets of finite ordinals, then CRSF is exactly polynomial time.

Proof

P $\subseteq$ **CRSF**: We use the lemma, plus ordinal arithmetic to compute the bounds on time and tape length.

CRSF $\subseteq$ **P**: Every initial CRSF function on sets corresponds to a simple function on their Mostowski graphs, and it is straightforward to manipulate graphs in P. We can also simulate recursion in P, and the subset bound stops this from blowing up the size of graphs too much.

Infinite time Turing machines

These are defined as finite machines, except

- we replace the tape length n with ω
- we replace the time t with an ordinal τ
- we put in rules for what happens at a time which is a limit ordinal; in particular, each cell of the tape contains the lim sup of the contents at previous times.

Infinite time Turing machines

These are defined as finite machines, except

- we replace the tape length n with ω
- we replace the time t with an ordinal τ
- we put in rules for what happens at a time which is a limit ordinal; in particular, each cell of the tape contains the lim sup of the contents at previous times.

Polynomial time now means time ω^k for some $k < \omega$.

Infinite time Turing machines

Theorem

If we restrict the input and output to infinite subsets of ω, then CRSF contains exactly the functions computed by polynomial time ITTMs.

Proof

P $\subseteq$ **CRSF**: As before, with tweaks for the limit case.

CRSF $\subseteq$ **P**: This is more complicated, but the same in spirit as before. We use a result of Friedman and Welch characterizing P on ITTMs as functions uniformly definable in the L hierarchy.

Circuits in complexity theory

Circuits in complexity theory

A finite (binary, DeMorgan) circuit is a directed acyclic graph with

- source nodes labelled as inputs, or as constants 0 or 1
- possibly some nodes labelled as outputs
- internal nodes labelled as $\bigwedge$, $\bigvee$ or $\neg$ gates
  where $\neg$ gates have exactly one incoming edge.

Circuits in complexity theory

If we have

- input nodes labelled $x_0, \ldots, x_{n-1}$
- output nodes labelled $y_0, \ldots, y_{m-1}$

then the circuit computes a function from $\{0,1\}^n$ to $\{0,1\}^m$

Circuits in complexity theory

If we have

- input nodes labelled $x_0, \ldots, x_{n-1}$
- output nodes labelled $y_0, \ldots, y_{m-1}$

then the circuit computes a function from $\{0,1\}^n$ to $\{0,1\}^m$
— or an integer function, with domain $[0, 2^n)$ and range $[0, 2^m)$.

Circuits in complexity theory

If we have

- input nodes labelled $x_0, \ldots, x_{n-1}$
- output nodes labelled $y_0, \ldots, y_{m-1}$

then the circuit computes a function from $\{0,1\}^n$ to $\{0,1\}^m$
— or an integer function, with domain $[0, 2^n)$ and range $[0, 2^m)$.

To compute a function F with domain $\{0,1\}^*$ we use a family of circuits $(C_n)_{n\in\omega}$ where C_n computes F restricted to $\{0,1\}^n$.

Circuits in complexity theory

If we have

- input nodes labelled $x_0, \ldots, x_{n-1}$
- output nodes labelled $y_0, \ldots, y_{m-1}$

then the circuit computes a function from $\{0,1\}^n$ to $\{0,1\}^m$
— or an integer function, with domain $[0, 2^n)$ and range $[0, 2^m)$.

To compute a function F with domain $\{0,1\}^*$ we use a family of circuits $(C_n)_{n\in\omega}$ where C_n computes F restricted to $\{0,1\}^n$.

F has *polynomial size circuits* if $|C_n| \leq p(n)$ for some polynomial p.

Circuits in complexity theory

If we have

- input nodes labelled $x_0, \ldots, x_{n-1}$
- output nodes labelled $y_0, \ldots, y_{m-1}$

then the circuit computes a function from $\{0,1\}^n$ to $\{0,1\}^m$
— or an integer function, with domain $[0, 2^n)$ and range $[0, 2^m)$.

To compute a function F with domain $\{0,1\}^*$ we use a family of circuits $(C_n)_{n\in\omega}$ where C_n computes F restricted to $\{0,1\}^n$.

F has *polynomial size circuits* if $|C_n| \leq p(n)$ for some polynomial p.

Theorem

The polynomial time functions are exactly those with uniform polynomial size circuits.

Infinite circuits

A circuit is a well-founded directed graph with

- source nodes labelled as inputs (or as constants 0 or 1)
- possibly some nodes labelled as outputs
- internal nodes labelled as $\bigwedge$, $\bigvee$ or $\neg$ gates.

Here $\bigwedge$ and $\bigvee$ gates may have infinitely many incoming edges.
$\neg$ gates have exactly one incoming edge.

Infinite circuits

A circuit is a well-founded directed graph with

- source nodes labelled as inputs (or as constants 0 or 1)
- possibly some nodes labelled as outputs
- internal nodes labelled as $\bigwedge$, $\bigvee$ or $\neg$ gates.

Here $\bigwedge$ and $\bigvee$ gates may have infinitely many incoming edges.
$\neg$ gates have exactly one incoming edge.

By well-foundedness, for any $0/1$ assignment to the input nodes, there is a unique *computation* of the circuit — that is, a $0/1$ assignment to all nodes.

Infinite circuits

A circuit is a well-founded directed graph with

- source nodes labelled as inputs (or as constants 0 or 1)
- possibly some nodes labelled as outputs
- internal nodes labelled as $\bigwedge$, $\bigvee$ or $\neg$ gates.

Here $\bigwedge$ and $\bigvee$ gates may have infinitely many incoming edges.
$\neg$ gates have exactly one incoming edge.

By well-foundedness, for any $0/1$ assignment to the input nodes, there is a unique *computation* of the circuit — that is, a $0/1$ assignment to all nodes.

. . . but it may require strong recursion to construct the computation.

Some notation

We write $<$ for the order induced by $\in$.

That is, $a < b$ if and only if $a \in \mathrm{tc}(b)$.

Some notation

We write $<$ for the order induced by $\in$.
That is, $a < b$ if and only if $a \in \mathrm{tc}(b)$.

We write $[b]$ for the set $\mathrm{tc}(\{b\})$.
Hence $[b] = \{a : a \leq b\}$.

Infinite circuits

Formally, an *embedded circuit* is a triple $\langle c, E, \lambda \rangle$ where

Infinite circuits

Formally, an *embedded circuit* is a triple $\langle c, E, \lambda \rangle$ where

- c is any set.
  The nodes of the circuit are the members of $[c]$.

Infinite circuits

Formally, an *embedded circuit* is a triple $\langle c, E, \lambda \rangle$ where

- c is any set.
  The nodes of the circuit are the members of $[c]$.
- $E \subseteq [c] \times [c]$ is the underlying graph of the circuit satisfying $\langle x, y \rangle \in E \to x < y$.

Infinite circuits

Formally, an *embedded circuit* is a triple $\langle c, E, \lambda \rangle$ where

- c is any set.
  The nodes of the circuit are the members of $[c]$.
- $E \subseteq [c] \times [c]$ is the underlying graph of the circuit satisfying $\langle x, y \rangle \in E \rightarrow x < y$.
- λ is a labelling function $[c] \rightarrow \{0, 1, *, \bigwedge, \bigvee, \neg\}$.
  The nodes labelled $*$ are input nodes
  Each node labelled $\neg$ has exactly one E-predecessor.

Infinite circuits

Formally, an *embedded circuit* is a triple $\langle c, E, \lambda \rangle$ where

- c is any set.
  The nodes of the circuit are the members of $[c]$.
- $E \subseteq [c] \times [c]$ is the underlying graph of the circuit satisfying $\langle x, y \rangle \in E \to x < y$.
- λ is a labelling function $[c] \to \{0, 1, *, \bigwedge, \bigvee, \neg\}$.
  The nodes labelled $*$ are input nodes
  Each node labelled $\neg$ has exactly one E-predecessor.

We say that the circuit has *size* c.

Infinite circuits

Formally, an *embedded circuit* is a triple $\langle c, E, \lambda \rangle$ where

- c is any set.
  The nodes of the circuit are the members of $[c]$.
- $E \subseteq [c] \times [c]$ is the underlying graph of the circuit satisfying $\langle x, y \rangle \in E \to x < y$.
- λ is a labelling function $[c] \to \{0, 1, *, \bigwedge, \bigvee, \neg\}$.
  The nodes labelled $*$ are input nodes
  Each node labelled $\neg$ has exactly one E-predecessor.

We say that the circuit has *size* c.

A computation is a correct assignment of $0/1$ values to all nodes.
We identify computations with subsets of $[c]$.

Infinite circuits

Formally, an *embedded circuit* is a triple $\langle c, E, \lambda \rangle$ where

- c is any set.
  The nodes of the circuit are the members of $[c]$.
- $E \subseteq [c] \times [c]$ is the underlying graph of the circuit satisfying $\langle x, y \rangle \in E \to x < y$.
- λ is a labelling function $[c] \to \{0, 1, *, \bigwedge, \bigvee, \neg\}$.
  The nodes labelled $*$ are input nodes
  Each node labelled $\neg$ has exactly one E-predecessor.

We say that the circuit has *size* c.

A computation is a correct assignment of $0/1$ values to all nodes.
We identify computations with subsets of $[c]$.

We can thus construct computations in CRSF using subset-bounded recursion.

Infinite circuits

That is, let $C = \langle c, E, \lambda \rangle$ be a circuit and let a be its set of input nodes.

Definition

Given any $A \subseteq a$, a *computation of C on A* is a subset $W \subseteq [c]$ which, informally, assigns $0/1$ values to the nodes of C using the usual rules of a Boolean circuit with input A.

That is, for all $u \in [c]$,

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

Strings

We would like circuits which can input and output arbitrary sets of a given “complexity”.

Strings

We would like circuits which can input and output arbitrary sets of a given "complexity".

Definition

For any set a, an *a-string* is just a subset of a.
We think of it as assigning a $0/1$ value to every member of a.

For example, for $n \in \omega$ we can identify the usual binary strings in $\{0, 1\}^n$ with n-strings.

Circuits with string inputs and outputs

Definition

An *embedded circuit with input size a and output size p* is a tuple $C = \langle c, E, \lambda, a, p, \mu, \nu \rangle$ where

- $\langle c, E, \lambda \rangle$ is a circuit

and we are also given

- A partial function $\mu : [c] \to a$
  mapping every input node to a member of a
- A function $\nu : p \to [c]$
  which maps every element of p to some node in $[c]$.

Circuits with string inputs and outputs

Definition

An *embedded circuit with input size a and output size p* is a tuple $C = \langle c, E, \lambda, a, p, \mu, \nu \rangle$ where

- $\langle c, E, \lambda \rangle$ is a circuit

and we are also given

- A partial function $\mu : [c] \to a$
  mapping every input node to a member of a
- A function $\nu : p \to [c]$
  which maps every element of p to some node in $[c]$.

This computes a function which takes as input an a-string A and outputs a p-string P.

Lemma

The function $\langle C, A \rangle \mapsto P$ is in CRSF.

Coding sets as strings

We know how to code natural numbers as finite binary strings which can then be processed by a circuit.

How do we code sets as strings?

Embedded Mostowski collapse

Recall that the Mostowski graph $\mathcal{G}(a)$ of a set a is

- well-founded
- extensional
- "accessible pointed with sink a"
  – that is, there is a path from every node to a.

Embedded Mostowski collapse

Recall that the Mostowski graph $\mathcal{G}(a)$ of a set a is

- well-founded
- extensional
- "accessible pointed with sink a"
  – that is, there is a path from every node to a.

On the other hand, any well-founded, extensional, accessible pointed graph is isomorphic to the Mostowski graph of some set, which we call its Mostowski collapse.

Initial functions - embedded Mostowski collapse

Recall $x < y$ means $x \in \mathrm{tc}(y)$ and $[a] = \mathrm{tc}(\{a\})$.
We write $[a]^2$ for $[a] \times [a]$.

Definition

A *diagram* is a pair $\langle a, E\rangle$ of sets such that

$$\langle x, y\rangle \in E \to x < y.$$

Initial functions - embedded Mostowski collapse

Recall $x < y$ means $x \in \mathrm{tc}(y)$ and $[a] = \mathrm{tc}(\{a\})$.
We write $[a]^2$ for $[a] \times [a]$.

Definition

A *diagram* is a pair $\langle a, E \rangle$ of sets such that

$$\langle x, y \rangle \in E \to x < y.$$

A diagram represents the graph with nodes $[a]$ and edges $E \cap [a]^2$.

This graph must be well-founded, because $\mathcal{G}(a)$ is.
In general it is not extensional or accessible pointed. Nevertheless, we can define its Mostowski collapse.

Initial functions - embedded Mostowski collapse

Definition

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

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

Initial functions - embedded Mostowski collapse

Definition

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

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

Example: $M(4, \{\langle 3, 4\rangle, \langle 2, 3\rangle, \langle 0, 3\rangle\}) = \{1\}$.

Initial functions - embedded Mostowski collapse

Definition

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

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

Example: $M(4, \{\langle 3, 4\rangle, \langle 2, 3\rangle, \langle 0, 3\rangle\}) = \{1\}$.

Notice:

- this is definable by a simple $\in$-recursion
- $|M(a, E)| \leq |a|$
- $\text{rank}(M(a, E)) \leq \text{rank}(a)$.

Coding

Definition

For sets a, b we say that *b is embeddable in a* if

$$b = M(a, E) \text{ for some } E.$$

Coding

Definition

For sets a, b we say that *b is embeddable in a* if

$$b = M(a, E) \text{ for some } E.$$

Example

Any set a is embeddable in itself:
we have $a = M(a, E)$ where $E = \{\langle x, y\rangle \in [a]^2 : x \in y\}$.

Coding

Definition

For sets a, b we say that b *is embeddable in* a if

$$b = M(a, E) \text{ for some } E.$$

Example

Any set a is embeddable in itself:
we have $a = M(a, E)$ where $E = \{\langle x, y\rangle \in [a]^2 : x \in y\}$.

We think of such a b as having "size" a and being coded by E.
Without loss of generality, we may assume E is an $[a]^2$-string.

We can use E as the input or output of a circuit, which can then compute things about the set b.

Circuits computing set functions

Definition

Let f be a function from sets to sets. Let a, p be sets.

Let C be an embedded circuit with input size $[a]^2$ and output size $[p]^2$ such that for every $[a]^2$-string E,

$$M(p, C(E)) = f(M(a, E))$$

where $C(E)$ is the $[p]^2$-string output by C on input E.

Then we say that C *computes* f *on sets embeddable in* a.

Circuits computing set functions

Definition

Let f be a function from sets to sets. Let a, p be sets.

Let C be an embedded circuit with input size $[a]^2$ and output size $[p]^2$ such that for every $[a]^2$-string E,

$$M(p, C(E)) = f(M(a, E))$$

where $C(E)$ is the $[p]^2$-string output by C on input E.

Then we say that C *computes* f *on sets embeddable in* a.

The definition extends naturally to functions with many inputs.

Small circuits

Recall that the *size* of a circuit $\langle c, E, \lambda, a, p, \mu, \nu \rangle$ is the set c.

Definition

A function f from sets to sets *has small circuits* if there is a family C_a of circuits, parametrized by a set a, such that for every a

- C_a computes f on sets embeddable in a.
- C_a has size $s(a)$
- C_a has output size $t(a)$

where s and t are smash-terms.

Definition

Smash-terms are functions built from variables, the constant 0 and the functions pair, $\times$, tc, $\odot$ and $\#$.

They play the role that polynomials usually play in complexity theory.

Small circuits for CRSF

Theorem

Every CRSF function has small circuits.

Small circuits for CRSF

Theorem

Every CRSF function has small circuits.

Proof

Recall that the initial functions of CRSF are: 0, projection, pairing, union, conditional, transitive closure, cartesian product, $\odot$, $\#$, M.

Small circuits for CRSF

Theorem

Every CRSF function has small circuits.

Proof

Recall that the initial functions of CRSF are: 0, projection, pairing, union, conditional, transitive closure, cartesian product, $\odot$, $\#$, M.

These correspond to straightforward operations on diagrams.

Proof continued

Example: small circuits for union

Proof continued

Example: small circuits for union

Given a and E, consider the set

$$F = \left\{ \langle x, y \rangle \in [a]^2 : \begin{array}{ll} \langle x, y \rangle \in E & \text{if } y \neq a \\ \exists z \in [a], \langle x, z \rangle \in E \wedge \langle z, a \rangle \in E & \text{if } y = a \end{array} \right\}.$$

Then $M(a, F) = \bigcup M(a, E)$.

Proof continued

Example: small circuits for union

Given a and E, consider the set

$$F = \left\{ \langle x, y \rangle \in [a]^2 : \begin{array}{ll} \langle x, y \rangle \in E & \text{if } y \neq a \\ \exists z \in [a], \langle x, z \rangle \in E \wedge \langle z, a \rangle \in E & \text{if } y = a \end{array} \right\}.$$

Then $M(a, F) = \bigcup M(a, E)$.

This formula can easily be made into a small circuit family C_a, where each C_a takes an $[a]^2$-string E as input and outputs the $[a]^2$-string F.

Proof continued

Closure under composition is straightforward.

Proof continued

For closure under subset bounded recursion, suppose

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

and we have small circuits for g and h.

Proof continued

For closure under subset bounded recursion, suppose

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

and we have small circuits for g and h.

We first construct circuits C_b for the function $g(b, S) \cap h(b)$.

We may assume all circuits C_b have the same fixed size $t(a)$, where t is a smash-term. In other words, the underlying graph of every C_b is the Mostowski graph $\mathcal{G}(t(a))$ of $t(a)$.

Proof continued

For closure under subset bounded recursion, suppose

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

and we have small circuits for g and h.

We first construct circuits C_b for the function $g(b, S) \cap h(b)$.

We may assume all circuits C_b have the same fixed size $t(a)$, where t is a smash-term. In other words, the underlying graph of every C_b is the Mostowski graph $\mathcal{G}(t(a))$ of $t(a)$.

To compute $f(a)$, we simulate the recursion. We do this with a circuit, of size $a \# t(a)$, constructed as follows:

- take the graph $\mathcal{G}(a)$
- replace each node b with a copy of the circuit C_b and add suitable wires. $\square$

Uniform small circuits

The circuits given by the theorem are highly uniform.

In particular, the circuit C_a computing f on sets embeddable in a can be computed from a by a CRSF function.

Uniform small circuits

The circuits given by the theorem are highly uniform.

In particular, the circuit C_a computing f on sets embeddable in a can be computed from a by a CRSF function.

Corollary

The CRSF functions are exactly the functions from sets to sets computable by uniform families of small circuits.

Circuit lower bounds

Theorem [Hastad '89]

There is no family of finite circuits with number of nodes polynomial in n and depth polynomial in $\log(\log n)$ which calculates the parity of n-bit strings.

Circuit lower bounds

Corollary

There is no function f in CRSF such that $f(a)$ is the parity of $|a|$ for every finite a.

Circuit lower bounds

Corollary

There is no function f in CRSF such that $f(a)$ is the parity of $|a|$ for every finite a.

Proof

Such an f would have circuits C_a of size some smash-term $t(a)$.

Circuit lower bounds

Corollary

There is no function f in CRSF such that $f(a)$ is the parity of $|a|$ for every finite a.

Proof

Such an f would have circuits C_a of size some smash-term $t(a)$.
Choose $n \in \omega$ of the form 2^{2^k} and let $a = \mathbb{P}(\mathbb{P}(k))$.
Then $\mathrm{rank}(a) = k + 2$, $|a| = n$ and $|[a]| \leq 2n$.

Circuit lower bounds

Corollary

There is no function f in CRSF such that $f(a)$ is the parity of $|a|$ for every finite a.

Proof

Such an f would have circuits C_a of size some smash-term $t(a)$.
Choose $n \in \omega$ of the form 2^{2^k} and let $a = \mathbb{P}(\mathbb{P}(k))$.
Then $\mathrm{rank}(a) = k + 2$, $|a| = n$ and $|[a]| \leq 2n$.

Smash terms increase rank and cardinality of transitive closure by at most a polynomial amount. Therefore C_a has depth polynomial in $k = \log(\log n)$ and number of nodes polynomial in n.

Circuit lower bounds

Corollary

There is no function f in CRSF such that $f(a)$ is the parity of $|a|$ for every finite a.

Proof

Such an f would have circuits C_a of size some smash-term $t(a)$.
Choose $n \in \omega$ of the form 2^{2^k} and let $a = \mathbb{P}(\mathbb{P}(k))$.
Then $\mathrm{rank}(a) = k + 2$, $|a| = n$ and $|[a]| \leq 2n$.

Smash terms increase rank and cardinality of transitive closure by at most a polynomial amount. Therefore C_a has depth polynomial in $k =$log(log n) and number of nodes polynomial in n.

We can use C_a to compute the parity of subsets of a, and therefore of subsets of n. This is a contradiction.

P vs NP on sets

There is a natural analogue of NP for CRSF, namely relations of the form

$$\varphi(a) \longleftrightarrow \exists E \text{ a } t(a)\text{-string such that } \theta(a, E)$$

where t is a smash-term and θ is a CRSF relation.

P vs NP on sets

There is a natural analogue of NP for CRSF, namely relations of the form

$$\varphi(a) \longleftrightarrow \exists E \text{ a } t(a)\text{-string such that } \theta(a, E)$$

where t is a smash-term and θ is a CRSF relation.

This class is strictly bigger than CRSF. (This already follows indirectly from a result of Schindler, that P $\neq$ NP for infinite time Turing machines.)

P vs NP on sets

There is a natural analogue of NP for CRSF, namely relations of the form

$$\varphi(a) \longleftrightarrow \exists E \text{ a } t(a)\text{-string such that } \theta(a, E)$$

where t is a smash-term and θ is a CRSF relation.

This class is strictly bigger than CRSF. (This already follows indirectly from a result of Schindler, that P $\neq$ NP for infinite time Turing machines.)

We can show this directly, even for hereditarily finite a, by writing an NP expression for parity:

$|a|$ is even if and only if there exists an $a \times a$-string E giving a partition of a into disjoint pairs.