# Implicit Computational Complexity

**Guest Lecture for the course *Algorithms and Complexity***

Gianluca Curzi

School of Computer Science

University of Birmingham

01/04/2022

# Warm up

- **Question:** What is computational complexity?

# Implicit Computational Complexity

- **Answer to previous question:** computational complexity studies *complexity classes*, i.e. classes of languages (resp. functions) that can be accepted (resp. computed) by a machine (e.g. Turing machine) in a certain resource bound (e.g. time and space)
- *Implicit computational complexity* (*ICC*): branch of computational complexity describing complexity classes **without** explicit reference to machine models or cost bounds.
- ICC originates in the 90's with seminal paper on *safe recursion* [Bellantoni and Cook 92].

# Implicit Computational Complexity

- **Answer to previous question:** computational complexity studies *complexity classes*, i.e. classes of languages (resp. functions) that can be accepted (resp. computed) by a machine (e.g. Turing machine) in a certain resource bound (e.g. time and space)
- *Implicit computational complexity* (*ICC*): branch of computational complexity describing complexity classes **without** explicit reference to machine models or cost bounds.
- ICC originates in the 90's with seminal paper on *safe recursion* [Bellantoni and Cook 92].

# A closer look at ICC

- Borrows techniques and results from Mathematical Logic:
  - Recursion Theory (Restriction of primitive recursion schema);
  - Proof Theory (Curry-Howard correspondence);
  - Model Theory (Finite model theory).
- **One of the goals:** It aims to define programming language tools (e.g., type-systems) where runtime of programs can be statically certified.
- Pervasive notion of **stratification**: data are organized into *strata* (Bellantoni's safe recursion [Bellantoni and Cook 92], Leivant's predicative/ramified/tiered recursion [Leivant 95]).

# A closer look at ICC

- Borrows techniques and results from Mathematical Logic:
  - Recursion Theory (Restriction of primitive recursion schema);
  - Proof Theory (Curry-Howard correspondence);
  - Model Theory (Finite model theory).
- **One of the goals:** It aims to define programming language tools (e.g., type-systems) where runtime of programs can be statically certified.
- Pervasive notion of **stratification**: data are organized into *strata* (Bellantoni's safe recursion [Bellantoni and Cook 92], Leivant's predicative/ramified/tiered recursion [Leivant 95]).

# A closer look at ICC

- Borrows techniques and results from Mathematical Logic:
  - Recursion Theory (Restriction of primitive recursion schema);
  - Proof Theory (Curry-Howard correspondence);
  - Model Theory (Finite model theory).
- **One of the goals:** It aims to define programming language tools (e.g., type-systems) where runtime of programs can be statically certified.
- Pervasive notion of **stratification**: data are organized into *strata* (Bellantoni's safe recursion [Bellantoni and Cook 92], Leivant's predicative/ramified/tiered recursion [Leivant 95]).

# Some caveats. . .

- This lecture is about FPTIME (= functions computed in polynomial time).
  - languages accepted vs functions computed
  - time complexity vs space complexity
  - linear, polynomial, exponential, . . .
- Recursion-theoretic approach to characterise FPTIME in the style of ICC:
  - Algebra of primitive recursive functions **PR**
  - **Problem:** find weaker algebra of functions $X \subsetneq \mathbf{PR}$ such that $X = \text{FPTIME}$
  - **Idea:** restrict primitive recursion scheme
- Stepwise approach:
  - From primitive recursion to bounded recursion (on notation) → machine-free
  - Safe recursion on notation → machine-free + bound-free = ICC

# Some caveats. . .

- This lecture is about FPTIME (= functions computed in polynomial time).
  - languages accepted vs functions computed
  - time complexity vs space complexity
  - linear, polynomial, exponential, . . .
- Recursion-theoretic approach to characterise FPTIME in the style of ICC:
  - Algebra of primitive recursive functions **PR**
  - **Problem:** find weaker algebra of functions $X \subsetneq \textbf{PR}$ such that $X = \text{FPTIME}$
  - **Idea:** restrict primitive recursion scheme
- Stepwise approach:
  - From primitive recursion to bounded recursion (on notation) $\rightarrow$ machine-free
  - Safe recursion on notation $\rightarrow$ machine-free + bound-free = ICC

# Some caveats...

- This lecture is about FPTIME (= functions computed in polynomial time).
  - languages accepted vs functions computed
  - time complexity vs space complexity
  - linear, polynomial, exponential, ...
- Recursion-theoretic approach to characterise FPTIME in the style of ICC:
  - Algebra of primitive recursive functions **PR**
  - **Problem:** find weaker algebra of functions $X \subsetneq$ **PR** such that $X =$ FPTIME
  - **Idea:** restrict primitive recursion scheme
- Stepwise approach:
  - From primitive recursion to bounded recursion (on notation) $\rightarrow$ machine-free
  - Safe recursion on notation $\rightarrow$ machine-free + bound-free = ICC

# A recap of primitive recursive functions

**PR** is the smallest class of number-theoretic functions such that:

- It contains the *basic functions*
  - Constant zero: $0 \in \mathbb{N}$
  - Successor: $S : \mathbb{N} \to \mathbb{N}$, $S(x) = x + 1$
  - Projections: for any $k \in \mathbb{N}$ and $i \leq k$, $\pi_i^k : \mathbb{N}^k \to \mathbb{N}$, $\pi_i^k(x_1, \ldots, x_k) = x_i$
- It is closed under the *composition scheme*:
  - from $h : \mathbb{N}^{n+1} \to \mathbb{N}$ and $g : \mathbb{N}^n \to \mathbb{N}$ define $f : \mathbb{N}^n \to \mathbb{N}$ such that:

$$f(\vec{x}) = h(\vec{x}, g(\vec{x}))$$

- It is closed under the *primitive recursion scheme*:
  - from $g : \mathbb{N}^n \to \mathbb{N}$ to $h : \mathbb{N}^{n+2} \to \mathbb{N}$ define $f : \mathbb{N}^{n+1} \to \mathbb{N}$ such that:

$$\begin{aligned} f(0, \vec{y}) &= g(\vec{y}) \\ f(S(x), \vec{y}) &= h(x, \vec{y}, f(x, \vec{y})) \end{aligned}$$

# A recap of primitive recursive functions

**PR** is the smallest class of number-theoretic functions such that:

- It contains the *basic functions*
  - Constant zero: $0 \in \mathbb{N}$
  - Successor: $S : \mathbb{N} \to \mathbb{N}$, $S(x) = x + 1$
  - Projections: for any $k \in \mathbb{N}$ and $i \leq k$, $\pi_i^k : \mathbb{N}^k \to \mathbb{N}$, $\pi_i^k(x_1, \ldots, x_k) = x_i$
- It is closed under the *composition scheme*:
  - from $h : \mathbb{N}^{n+1} \to \mathbb{N}$ and $g : \mathbb{N}^n \to \mathbb{N}$ define $f : \mathbb{N}^n \to \mathbb{N}$ such that:

$$f(\vec{x}) = h(\vec{x}, g(\vec{x}))$$

- It is closed under the *primitive recursion scheme*:
  - from $g : \mathbb{N}^n \to \mathbb{N}$ to $h : \mathbb{N}^{n+2} \to \mathbb{N}$ define $f : \mathbb{N}^{n+1} \to \mathbb{N}$ such that:

$$\begin{aligned} f(0, \vec{y}) &= g(\vec{y}) \\ f(S(x), \vec{y}) &= h(x, \vec{y}, f(x, \vec{y})) \end{aligned}$$

# A recap of primitive recursive functions

**PR** is the smallest class of number-theoretic functions such that:

- It contains the *basic functions*
  - Constant zero: $0 \in \mathbb{N}$
  - Successor: $S : \mathbb{N} \to \mathbb{N}$, $S(x) = x + 1$
  - Projections: for any $k \in \mathbb{N}$ and $i \leq k$, $\pi_i^k : \mathbb{N}^k \to \mathbb{N}$, $\pi_i^k(x_1, \ldots, x_k) = x_i$
- It is closed under the *composition scheme*:
  - from $h : \mathbb{N}^{n+1} \to \mathbb{N}$ and $g : \mathbb{N}^n \to \mathbb{N}$ define $f : \mathbb{N}^n \to \mathbb{N}$ such that:

$$f(\vec{x}) = h(\vec{x}, g(\vec{x}))$$

- It is closed under the *primitive recursion scheme*:
  - from $g : \mathbb{N}^n \to \mathbb{N}$ to $h : \mathbb{N}^{n+2} \to \mathbb{N}$ define $f : \mathbb{N}^{n+1} \to \mathbb{N}$ such that:

$$\begin{aligned} f(0, \vec{y}) &= g(\vec{y}) \\ f(S(x), \vec{y}) &= h(x, \vec{y}, f(x, \vec{y})) \end{aligned}$$

# Recursive functions as a machine model

- Original goal: **extensional** definition as a class of functions
- Natural operational interpretation as **rewriting**
- **However:** no notion of constant time elementary step.
- Rewriting involves duplication of data of arbitrary size and of computations of arbitrary length.
- Need of non trivial data structures (stack) to (naïvely) implement primitive recursion.

# Recursive functions as a machine model

- Original goal: **extensional** definition as a class of functions
- Natural operational interpretation as **rewriting**
- **However:** no notion of constant time elementary step.
- Rewriting involves duplication of data of arbitrary size and of computations of arbitrary length.
- Need of non trivial data structures (stack) to (naïvely) implement primitive recursion.

# A notational problem

- Complexity classes defined for **binary** strings (e.g. 1001)
- Binary representation of natural numbers:

$$\begin{array}{rcl} n & \mapsto & |n| \\ 9 = ||||||||| & \mapsto & 1001 \end{array}$$

where $|n| \approx \log n$

- Usual recursion is on **unary** notation:

$$\text{linear in } n \quad = \quad \text{exponential in } |n|$$

indeed $n = 2^{\log n} \approx 2^{|n|}$

- **Solution:** recursion on **(binary) notation**.

# A notational problem

- Complexity classes defined for **binary** strings (e.g. 1001)
- Binary representation of natural numbers:

$$n \quad \mapsto \quad |n|$$

$$9 = ||||||||| \quad \mapsto \quad 1001$$

where $|n| \approx \log n$

- Usual recursion is on **unary** notation:

$$\text{linear in } n \quad = \quad \text{exponential in } |n|$$

indeed $n = 2^{\log n} \approx 2^{|n|}$

- **Solution:** recursion on **(binary) notation**.

# Recursion on notation

- Data: natural numbers
- Two "successors":
  - $s_0(n) = 2n$ (i.e. adding 0 at the least significant position)
  - $s_1(n) = 2n + 1$ (i.e. adding 1 at the least significant position)
- *Recursion on notation*:
  - from $g : \mathbb{N}^n \to \mathbb{N}$ and $h_0, h_1 : \mathbb{N}^{n+2} \to \mathbb{N}$ define $f : \mathbb{N}^n \to \mathbb{N}$ such that:

$$\begin{aligned} f(0, \vec{y}) &= g(\vec{y}) \\ f(s_0(x), \vec{y}) &= h_0(x, \vec{y}, f(x, \vec{y})) \qquad x \neq 0 \\ f(s_1(x), \vec{y}) &= h_1(x, \vec{y}, f(x, \vec{y})) \end{aligned}$$

- Now recursion converges **quickly** to a base case: $f(n)$ involves at most $\log n \approx |n|$ recursive calls.

# Recursion on notation

- Data: natural numbers
- Two "successors":
  - $s_0(n) = 2n$ (i.e. adding 0 at the least significant position)
  - $s_1(n) = 2n + 1$ (i.e. adding 1 at the least significant position)
- *Recursion on notation*:
  - from $g : \mathbb{N}^n \to \mathbb{N}$ and $h_0, h_1 : \mathbb{N}^{n+2} \to \mathbb{N}$ define $f : \mathbb{N}^n \to \mathbb{N}$ such that:

$$\begin{array}{rcll} f(0, \vec{y}) & = & g(\vec{y}) & \\ f(s_0(x), \vec{y}) & = & h_0(x, \vec{y}, f(x, \vec{y})) & \quad x \neq 0 \\ f(s_1(x), \vec{y}) & = & h_1(x, \vec{y}, f(x, \vec{y})) & \end{array}$$

- Now recursion converges **quickly** to a base case: $f(n)$ involves at most $\log n \approx |n|$ recursive calls.

# Recursion on notation is too generous

- Function double($x$) such that $|\text{double}(x)| = 2 \cdot |x|$:

$$\begin{aligned} \text{double}(0) &= 1 \\ \text{double}(s_0(x)) &= s_0(s_0(\text{double}(x))) \qquad x \neq 0 \\ \text{double}(s_1(x)) &= s_0(s_0(\text{double}(x))) \end{aligned}$$

- Function exp($x$):

$$\begin{aligned} \exp(0) &= 1 \\ \exp(s_0(x)) &= \text{double}(\exp(x)) \qquad x \neq 0 \\ \exp(s_1(x)) &= \text{double}(\exp(x)) \end{aligned}$$

- $\exp(x)$ has exponential length in $|x|$, i.e. $|\exp(x)| = 2^{|x|}$.

# Recursion on notation is too generous

- Function double($x$) such that $|\text{double}(x)| = 2 \cdot |x|$:

$$
\begin{array}{rcll}
\text{double}(0) & = & 1 & \\
\text{double}(s_0(x)) & = & s_0(s_0(\text{double}(x))) & \quad x \neq 0 \\
\text{double}(s_1(x)) & = & s_0(s_0(\text{double}(x))) &
\end{array}
$$

- Function exp($x$):

$$
\begin{array}{rcll}
\exp(0) & = & 1 & \\
\exp(s_0(x)) & = & \text{double}(\exp(x)) & \quad x \neq 0 \\
\exp(s_1(x)) & = & \text{double}(\exp(x)) &
\end{array}
$$

- $\exp(x)$ has exponential length in $|x|$, i.e. $|\exp(x)| = 2^{|x|}$.

# Bounded recursion on notation (Cobham 1965)

- *Bounded recursion on notation*:
  - from $g : \mathbb{N}^n \to \mathbb{N}$ and $h_0, h_1 : \mathbb{N}^{n+2} \to \mathbb{N}$ and $k : \mathbb{N}^{n+1} \to \mathbb{N}$

$$\begin{aligned} f(0, \vec{y}) &= g(\vec{y}) \\ f(s_0(x), \vec{y}) &= h_0(x, \vec{y}, f(x, \vec{y})) \qquad x \neq 0 \\ f(s_1(x), \vec{y}) &= h_1(x, \vec{y}, f(x, \vec{y})) \end{aligned}$$

  provided $f(x, \vec{y}) \leq k(x, \vec{y})$.

- We need an extra basic function to achieve the desired growth rate:

$$x \sharp y = 2^{|x| \cdot |y|}$$

**BRN** is the smallest class of number-theoretic functions such that:

- It contains the basic functions (zero, successor, projections) and *smash function*.
- It is closed under the composition scheme.
- It is closed under the *bounded recursion on notation* scheme.

# Bounded recursion on notation (Cobham 1965)

- *Bounded recursion on notation*:
  - from $g : \mathbb{N}^n \to \mathbb{N}$ and $h_0, h_1 : \mathbb{N}^{n+2} \to \mathbb{N}$ and $k : \mathbb{N}^{n+1} \to \mathbb{N}$

$$\begin{array}{rcll} f(0, \vec{y}) & = & g(\vec{y}) & \\ f(s_0(x), \vec{y}) & = & h_0(x, \vec{y}, f(x, \vec{y})) & \quad x \neq 0 \\ f(s_1(x), \vec{y}) & = & h_1(x, \vec{y}, f(x, \vec{y})) & \end{array}$$

  provided $f(x, \vec{y}) \leq k(x, \vec{y})$.

- We need an extra basic function to achieve the desired growth rate:

$$x \sharp y = 2^{|x| \cdot |y|}$$

**BRN** is the smallest class of number-theoretic functions such that:

- It contains the basic functions (zero, successor, projections) and *smash function*.
- It is closed under the composition scheme.
- It is closed under the *bounded recursion on notation* scheme.

# **BRN** is an algebra of polytime computable functions

**Theorem (Cobham, 65)**

**BRN** = FPTIME.

- FPTIME ⊆ **BRN**: Code TMs as functions of the algebra. The iteration of the transition function is representable because a priori polynomially bounded.
- **BRN** ⊆ FPTIME: By induction on the length of the definition, show that any function is computable by a polynomially bounded TM, exploiting the bound on the recursive definition.

# **BRN** is an algebra of polytime computable functions

**Theorem (Cobham, 65)**

**BRN** $=$ FPTIME.

- FPTIME $\subseteq$ **BRN**: Code TMs as functions of the algebra. The iteration of the transition function is representable because a priori polynomially bounded.
- **BRN** $\subseteq$ FPTIME: By induction on the length of the definition, show that any function is computable by a polynomially bounded TM, exploiting the bound on the recursive definition.

# **BRN** is an algebra of polytime computable functions

**Theorem (Cobham, 65)**

**BRN** = FPTIME.

- FPTIME $\subseteq$ **BRN**: Code TMs as functions of the algebra. The iteration of the transition function is representable because a priori polynomially bounded.
- **BRN** $\subseteq$ FPTIME: By induction on the length of the definition, show that any function is computable by a polynomially bounded TM, exploiting the bound on the recursive definition.

# A critique to Cobham characterization

- Cobham's paper is the birth of computational complexity as a respected theory, as it characterized FPTIME as a mathematically meaningful class.
- **However:** from the implicit computational complexity perspective, it is not as implicit as it seems:
  - It uses an explicit a priori bound on the construction
  - It "throws in" the polynomials (i.e., the $\sharp$ function) in the recipe, in order to make it work.
- We had to wait until the '90s to get a more "implicit" characterization of FPTIME...

# Safe recursion: idea

- Analysis of the exponential function exp($x$): recursive call of exp is in turn recursive parameter of double($x$).
- Different strategy to control the growth of function:

bounded recursion $\rightsquigarrow$ unbounded recursion + stratification (safe/normal)

- Function arguments are partitioned into normal and safe:

$$f(x_1, \ldots, x_n \,;\, y_1, \ldots, y_m)$$

- Safe recursion on notation:

$$\begin{aligned} f(0, \vec{x}; \vec{y}) &= g(\vec{x}; \vec{y}) \\ f(s_0 x, \vec{x}; \vec{y}) &= h_0(x, \vec{x}; \vec{y}, f(x, \vec{x}; \vec{y})) \\ f(s_1 x, \vec{x}; \vec{y}) &= h_1(x, \vec{x}; \vec{y}, f(x, \vec{x}; \vec{y})) \end{aligned}$$

- **Idea:** recursive call $f(x, \vec{x}; \vec{y})$ is never the recursive parameter of $h_i$.

# Safe recursion: idea

- Analysis of the exponential function $\exp(x)$: recursive call of exp is in turn recursive parameter of $\text{double}(x)$.
- Different strategy to control the growth of function:

bounded recursion $\rightsquigarrow$ unbounded recursion $+$ stratification (safe/normal)

- Function arguments are partitioned into normal and safe:

$$f(x_1, \ldots, x_n \,;\, y_1, \ldots, y_m)$$

- Safe recursion on notation:

$$\begin{array}{rcl} f(0, \vec{x}; \vec{y}) & = & g(\vec{x}; \vec{y}) \\ f(s_0 x, \vec{x}; \vec{y}) & = & h_0(x, \vec{x}; \vec{y}, f(x, \vec{x}; \vec{y})) \\ f(s_1 x, \vec{x}; \vec{y}) & = & h_1(x, \vec{x}; \vec{y}, f(x, \vec{x}; \vec{y})) \end{array}$$

- **Idea:** recursive call $f(x, \vec{x}; \vec{y})$ is never the recursive parameter of $h_i$.

# Safe composition

- Composition is constrained to respect this partition.
- Safe composition:

$$
\begin{aligned}
f(\vec{x};\vec{y}) &= h(\vec{x}; g(\vec{x};\vec{y})) \\
f(\vec{x};\vec{y}) &= h(g(\vec{x};);\vec{y}) \qquad \text{no safe parameters in g!}
\end{aligned}
$$

- **Idea:** We can move a normal argument in safe position but not vice versa:

$$
\begin{aligned}
h(x;y) \mapsto f(x;x): \quad & f(x;x) = h(x;\pi_1^1(x;)) = h(x;x) \\
h(x;y) \not\mapsto f(y;y): \quad & f(y;y) \neq h(\pi_1^1(;y);y) = h(y;y)
\end{aligned}
$$

# The algebra of function **BC**

**BC** is the smallest class of number-theoretic functions such that:

- It contains the following basic functions:
  - Constant zero: 0
  - Successors: $s_0(;x) = 2 \cdot x$ and $s_1(;x) = 2 \cdot x + 1$
  - Projections: $\pi^{n,m}_{i;}(x_1, \ldots, x_n; y_1, \ldots, y_m) = x_i$ and $\pi^{n,m}_{;j}(x_1, \ldots, x_n; y_1, \ldots, y_m) = y_j$
  - Predecessor: $P(;0) = 0$ and $P(;s_i(x)) = x$
  - Conditional:

$$C(;x,y,z) = \begin{cases} y & \text{if } x = s_0(x') \\ z & \text{if } x = s_1(x') \end{cases}$$

- It is closed under *safe recursion on notation* and *safe composition*.

# **BC** is an algebra of polytime computable functions

**Theorem (Bellantoni and Cook, 92)**

$f(\vec{x};) \in \mathbf{BC}$ iff $f(\vec{x}) \in \mathbf{BRN}$.

- if $f(\vec{x};) \in \mathbf{BC}$ then $f(\vec{x}) \in \mathbf{BRN}$:
  - For any $f(\vec{x};) \in \mathbf{BC}$ there is a polynomial $q_f$ such that

    $$|f(\vec{x};\vec{y})| \leq q_f(|\vec{x}|) + \max(|\vec{y}|) \qquad q_f \text{ polynomial}$$

  - Observe that such $q_f$ are definable in **BRN**
  - Thus, *safe recursion on notation* instance of *bounded recursion on notation*.
- If $f(\vec{x}) \in \mathrm{BRN}$ then $f(\vec{x};) \in \mathbf{BC}$.
  - By induction on derivation on Cobham's system, show that for any $f(\vec{x}) \in \mathbf{BRN}$ there is a function $h(w;\vec{x}) \in \mathbf{BC}$ and a polynomial $p_f$ such that $h(w;\vec{x}) = f(\vec{x})$ for all $\vec{x}$ and for any $w \geq p_f(|\vec{x}|)$
  - Now construct a function $b(\vec{x}) \in \mathbf{BC}$ such that $b(\vec{x};) \geq p_f(|\vec{x}|)$
  - Set $f(\vec{x};) = h(b(\vec{x};);\vec{x}) \in \mathbf{BC}$.

# **BC** is an algebra of polytime computable functions

**Theorem (Bellantoni and Cook, 92)**

$f(\vec{x};) \in \mathbf{BC}$ iff $f(\vec{x}) \in \mathbf{BRN}$.

- if $f(\vec{x};) \in \mathbf{BC}$ then $f(\vec{x}) \in \mathbf{BRN}$:
  - For any $f(\vec{x};) \in \mathbf{BC}$ there is a polynomial $q_f$ such that

$$|f(\vec{x};\vec{y})| \leq q_f(|\vec{x}|) + \max(|\vec{y}|) \qquad q_f \text{ polynomial}$$

  - Observe that such $q_f$ are definable in **BRN**
  - Thus, *safe recursion on notation* instance of *bounded recursion on notation*.
- If $f(\vec{x}) \in \mathrm{BRN}$ then $f(\vec{x};) \in \mathbf{BC}$.
  - By induction on derivation on Cobham's system, show that for any $f(\vec{x}) \in \mathbf{BRN}$ there is a function $h(w;\vec{x}) \in \mathbf{BC}$ and a polynomial $p_f$ such that $h(w;\vec{x}) = f(\vec{x})$ for all $\vec{x}$ and for any $w \geq p_f(|\vec{x}|)$
  - Now construct a function $b(\vec{x}) \in \mathbf{BC}$ such that $b(\vec{x};) \geq p_f(|\vec{x}|)$
  - Set $f(\vec{x};) = h(b(\vec{x};);\vec{x}) \in \mathbf{BC}$.

# **BC** is an algebra of polytime computable functions

**Theorem (Bellantoni and Cook, 92)**

$f(\vec{x};) \in \mathbf{BC}$ iff $f(\vec{x}) \in \mathbf{BRN}$.

- if $f(\vec{x};) \in \mathbf{BC}$ then $f(\vec{x}) \in \mathbf{BRN}$:
  - For any $f(\vec{x};) \in \mathbf{BC}$ there is a polynomial $q_f$ such that

    $$|f(\vec{x};\vec{y})| \leq q_f(|\vec{x}|) + \max(|\vec{y}|) \qquad q_f \text{ polynomial}$$

  - Observe that such $q_f$ are definable in **BRN**
  - Thus, *safe recursion on notation* instance of *bounded recursion on notation*.

- If $f(\vec{x}) \in \text{BRN}$ then $f(\vec{x};) \in \mathbf{BC}$.
  - By induction on derivation on Cobham's system, show that for any $f(\vec{x}) \in \mathbf{BRN}$ there is a function $h(w;\vec{x}) \in \mathbf{BC}$ and a polynomial $p_f$ such that $h(w;\vec{x}) = f(\vec{x})$ for all $\vec{x}$ and for any $w \geq p_f(|\vec{x}|)$
  - Now construct a function $b(\vec{x}) \in \mathbf{BC}$ such that $b(\vec{x};) \geq p_f(|\vec{x}|)$
  - Set $f(\vec{x};) = h(b(\vec{x};);\vec{x}) \in \mathbf{BC}$.

Thank you!
Questions?