A Rewriting Characterization of Higher-Order Feasibility via Tuple Interpretations

Ongoing joint work with Patrick Baillot, Ugo dal Lago,
Cynthia Kop, and **Deivid Vale**
June 8, 2022

Summary

Higher-order Feasibility

HO Rewriting and Tuple Interpretations

Runtime Complexity

BFFs Characterization

Outline

Higher-order Feasibility

HO Rewriting and Tuple Interpretations

Runtime Complexity

BFFs Characterization

Constable problem

Constable (1973) posed the problem of finding a **natural analogue** of polynomial time (P) for functionals of type:

$$(\mathbb{N} \to \mathbb{N})^k \times \mathbb{N}^\ell \to \mathbb{N}$$

This problem has been studied since the 70's.

Constable problem

Constable (1973) posed the problem of finding a **natural analogue** of polynomial time (P) for functionals of type:

$$(\mathbb{N} \to \mathbb{N})^k \times \mathbb{N}^\ell \to \mathbb{N}$$

This problem has been studied since the 70's.

Why this problem is interesting?

- most tasks considered feasible are in P

Constable problem

Constable (1973) posed the problem of finding a **natural analogue** of polynomial time (P) for functionals of type:

$$(\mathbb{N} \to \mathbb{N})^k \times \mathbb{N}^\ell \to \mathbb{N}$$

This problem has been studied since the 70's.

Why this problem is interesting?

- most tasks considered feasible are in P
- most tasks **outside of** P seems quite infeasible

Constable problem

Constable (1973) posed the problem of finding a **natural analogue** of polynomial time (P) for functionals of type:

$$(\mathbb{N} \to \mathbb{N})^k \times \mathbb{N}^\ell \to \mathbb{N}$$

This problem has been studied since the 70's.

Why this problem is interesting?

- most tasks considered feasible are in P
- most tasks **outside of** P seems quite infeasible
- almost all **reasonable** models of deterministic computation are **polynomially** related

Constable problem

Constable (1973) posed the problem of finding a **natural analogue** of polynomial time (P) for functionals of type:

$$(\mathbb{N} \to \mathbb{N})^k \times \mathbb{N}^\ell \to \mathbb{N}$$

This problem has been studied since the 70's.

Why this problem is interesting?

- most tasks considered feasible are in P
- most tasks **outside of** P seems quite infeasible
- almost all **reasonable** models of deterministic computation are **polynomially** related
- both P and PF have good closure properties

Basic Feasible Functionals (BFFs)

Good candidate? Let's bring... BFFs

Basic Feasible Functionals (BFFs)

Good candidate? Let's bring... BFFs

they are...

- second order functionals (Type-2)

Basic Feasible Functionals (BFFs)

Good candidate? Let's bring... BFFs

they are...

- second order functionals (Type-2)
- can be captured by type-2 limited recursion on notation

Basic Feasible Functionals (BFFs)

Good candidate? Let's bring... BFFs

they are...

- second order functionals (Type-2)
- can be captured by type-2 limited recursion on notation
- can be computed in terms of OTM in polynomial time

Basic Feasible Functionals (BFFs)

The BFF **recursive scheme**.

F is defined from G, H, and K by limited recursion on notation **(LRN)** if for all $\vec{f}, \vec{x}$, and y,

$$
\begin{aligned}
F(\vec{f}, \vec{x}, 0) &= G(\vec{f}, \vec{x}) \\
F(\vec{f}, \vec{x}, y) &= H(\vec{f}, \vec{x}, y, F(\vec{f}, \vec{x}, \lfloor x/2 \rfloor)), y > 0, \\
|F(\vec{f}, \vec{x}, y)| &\leq |K(\vec{f}, \vec{x}, y)|.
\end{aligned}
$$

Definition

The class **BFF** is the smallest class of functionals containing `FPTIME` and the application functional ($\lambda Fx.F(x)$), and closed under: **composition**, **expansion**, and **LRN**.

Goal

Our goal is to characterize **BFFs** via **higher-order rewriting** and **tuple interpretations**.

Outline

Higher-order Feasibility

HO Rewriting and Tuple Interpretations

Runtime Complexity

BFFs Characterization

Higher-Order Rewriting

Basic Idea: A form of typed lambda-calculus with function symbols and rules.

- abstraction and application

Higher-Order Rewriting

Basic Idea: A form of typed lambda-calculus with function symbols and rules.

- abstraction and application
- function symbols with arity:

$$
\begin{array}{rclrcl}
\mathsf{nil} & :: & \mathsf{list} & \texttt{cons} & :: & \mathsf{nat} \times \mathsf{list} \Longrightarrow \mathsf{natlist} \\
\texttt{map} & :: & (\mathsf{nat} \Longrightarrow \mathsf{nat}) \times \mathsf{list} \Longrightarrow \mathsf{list} & & &
\end{array}
$$

Higher-Order Rewriting

Basic Idea: A form of typed lambda-calculus with function symbols and rules.

- abstraction and application
- function symbols with arity:

$$\begin{array}{rclrcl} \mathsf{nil} & :: & \mathsf{list} & \texttt{cons} & :: & \mathsf{nat} \times \mathsf{list} \Longrightarrow \mathsf{natlist} \\ \texttt{map} & :: & \multicolumn{4}{l}{(\mathsf{nat} \Longrightarrow \mathsf{nat}) \times \mathsf{list} \Longrightarrow \mathsf{list}} \end{array}$$

- variables of higher-order type.

Strongly monotonic functionals in a nutshell

General idea:

- for every base type ι: let $(\!|\iota|\!) = \mathbb{N}^{p[\iota]}$ for some $p[\iota]$;
- say $\langle n_1, \dots, n_p \rangle > \langle m_1, \dots, m_p \rangle$ if $n_1 > m_1$ and each $n_i \geq m_i$;

- for each symbol $\mathtt{f} : [\sigma_1 \times \cdots \times \sigma_k] \Rightarrow \tau$: map $\mathtt{f}$ to a monotonic function in $(\!|\sigma_1|\!) \times \cdots \times (\!|\sigma_k|\!) \Rightarrow (\!|\tau|\!)$;
- prove that $[\![\ell]\!] > [\![r]\!]$ for all rules $\ell \to r$.

Strongly monotonic functionals in a nutshell

General idea:

- for every base type ι: let $(\!|\iota|\!) = \mathbb{N}^{p[\iota]}$ for some $p[\iota]$;
- say $\langle n_1, \dots, n_p \rangle > \langle m_1, \dots, m_p \rangle$ if $n_1 > m_1$ and each $n_i \geq m_i$;
- for every arrow type $\sigma \Rightarrow \tau$: let $(\!|\sigma \Rightarrow \tau|\!) = \{$ monotonic functions from $(\!|\sigma|\!)$ to $(\!|\tau|\!)$ $\}$

- for each symbol $\mathtt{f} : [\sigma_1 \times \cdots \times \sigma_k] \Rightarrow \tau$: map $\mathtt{f}$ to a monotonic function in $(\!|\sigma_1|\!) \times \cdots \times (\!|\sigma_k|\!) \Rightarrow (\!|\tau|\!)$;
- prove that $[\![\ell]\!] > [\![r]\!]$ for all rules $\ell \to r$.

Strongly monotonic functionals in a nutshell

General idea:

- for every base type ι: let $(\!|\iota|\!) = \mathbb{N}^{p[\iota]}$ for some $p[\iota]$;
- say $\langle n_1, \dots, n_p \rangle > \langle m_1, \dots, m_p \rangle$ if $n_1 > m_1$ and each $n_i \geq m_i$;
- for every arrow type $\sigma \Rightarrow \tau$: let $(\!|\sigma \Rightarrow \tau|\!) = \{$ monotonic functions from $(\!|\sigma|\!)$ to $(\!|\tau|\!)$ $\}$
- say $f > g$ if $f(x) > g(x)$ for all x
- for each symbol $\mathtt{f} : [\sigma_1 \times \cdots \times \sigma_k] \Rightarrow \tau$: map $\mathtt{f}$ to a monotonic function in $(\!|\sigma_1|\!) \times \cdots \times (\!|\sigma_k|\!) \Rightarrow (\!|\tau|\!)$;
- prove that $[\![\ell]\!] > [\![r]\!]$ for all rules $\ell \to r$.

Higher-order tuple interpretations: an example

$$
\begin{aligned}
\texttt{nil} &:: \mathsf{list} \\
\texttt{cons} &:: [\mathsf{nat} \times \mathsf{list}] \Rightarrow \mathsf{list} \\
\texttt{map} &:: [(\mathsf{nat} \Rightarrow \mathsf{nat}) \times \mathsf{list}] \Rightarrow \mathsf{list}
\end{aligned}
$$

$$
\begin{aligned}
\texttt{map}(F, \texttt{nil}) &\rightarrow \texttt{nil} \\
\texttt{map}(F, \texttt{cons}(x, a)) &\rightarrow \texttt{cons}(F \cdot x, \texttt{map}(F, a))
\end{aligned}
$$

Higher-order tuple interpretations: an example

$$\begin{array}{rcl} \texttt{nil} & :: & \text{list} \\ \texttt{cons} & :: & [\text{nat} \times \text{list}] \Rightarrow \text{list} \\ \texttt{map} & :: & [(\text{nat} \Rightarrow \text{nat}) \times \text{list}] \Rightarrow \text{list} \end{array}$$

$$\begin{array}{rcl} \texttt{map}(F, \texttt{nil}) & \to & \texttt{nil} \\ \texttt{map}(F, \texttt{cons}(x, a)) & \to & \texttt{cons}(F \cdot x, \texttt{map}(F, a)) \end{array}$$

Semantics: $(\!|\text{list}|\!) = \langle \text{cost}, \text{length}, \text{maximum} \rangle$

- $[\![\texttt{nil}]\!] = \langle 0, 0, 0 \rangle$
- $[\![\texttt{cons}(x, a)]\!] = \langle x_{\text{cost}} + a_{\text{cost}}, a_{\text{len}} + 1, \max(x_{\text{size}}, a_{\text{max}}) \rangle$
- $[\![\texttt{map}(F, a)]\!] = \langle \text{cost}, \text{length}, \text{maximum} \rangle$, where:
  - length:
  - maximum:
  - cost:

Higher-order tuple interpretations: an example

$$\begin{array}{rcl} \texttt{nil} & :: & \text{list} \\ \texttt{cons} & :: & [\text{nat} \times \text{list}] \Rightarrow \text{list} \\ \texttt{map} & :: & [(\text{nat} \Rightarrow \text{nat}) \times \text{list}] \Rightarrow \text{list} \end{array}$$

$$\begin{array}{rcl} \texttt{map}(F, \texttt{nil}) & \to & \texttt{nil} \\ \texttt{map}(F, \texttt{cons}(x, a)) & \to & \texttt{cons}(F \cdot x, \texttt{map}(F, a)) \end{array}$$

Semantics: $(\!|\text{list}|\!) = \langle \text{cost}, \text{length}, \text{maximum} \rangle$

- $[\![\texttt{nil}]\!] = \langle 0, 0, 0 \rangle$
- $[\![\texttt{cons}(x, a)]\!] = \langle x_{\text{cost}} + a_{\text{cost}}, a_{\text{len}} + 1, \max(x_{\text{size}}, a_{\max}) \rangle$
- $[\![\texttt{map}(F, a)]\!] = \langle \text{cost}, \text{length}, \text{maximum} \rangle$, where:
  - length: a_{len}
  - maximum:
  - cost:

Higher-order tuple interpretations: an example

$$\begin{array}{rcl} \texttt{nil} & :: & \mathsf{list} \\ \texttt{cons} & :: & [\mathsf{nat} \times \mathsf{list}] \Rightarrow \mathsf{list} \\ \texttt{map} & :: & [(\mathsf{nat} \Rightarrow \mathsf{nat}) \times \mathsf{list}] \Rightarrow \mathsf{list} \end{array}$$

$$\begin{array}{rcl} \texttt{map}(F, \texttt{nil}) & \to & \texttt{nil} \\ \texttt{map}(F, \texttt{cons}(x, a)) & \to & \texttt{cons}(F \cdot x, \texttt{map}(F, a)) \end{array}$$

Semantics: $(\!|\mathsf{list}|\!) = \langle \mathsf{cost}, \mathsf{length}, \mathsf{maximum} \rangle$

- $[\![\texttt{nil}]\!] = \langle 0, 0, 0 \rangle$
- $[\![\texttt{cons}(x, a)]\!] = \langle x_{\mathsf{cost}} + a_{\mathsf{cost}}, a_{\mathsf{len}} + 1, \max(x_{\mathsf{size}}, a_{\mathsf{max}}) \rangle$
- $[\![\texttt{map}(F, a)]\!] = \langle \mathsf{cost}, \mathsf{length}, \mathsf{maximum} \rangle$, where:
  - length: a_{len}
  - maximum: $F(\ \langle a_{\mathsf{cost}}, a_{\mathsf{max}} \rangle\ )_s$
  - cost:

Higher-order tuple interpretations: an example

$$\begin{array}{rcl} \texttt{nil} & :: & \text{list} \\ \texttt{cons} & :: & [\text{nat} \times \text{list}] \Rightarrow \text{list} \\ \texttt{map} & :: & [(\text{nat} \Rightarrow \text{nat}) \times \text{list}] \Rightarrow \text{list} \end{array}$$

$$\begin{array}{rcl} \texttt{map}(F, \texttt{nil}) & \to & \texttt{nil} \\ \texttt{map}(F, \texttt{cons}(x, a)) & \to & \texttt{cons}(F \cdot x, \texttt{map}(F, a)) \end{array}$$

Semantics: $(\!|\text{list}|\!) = \langle \text{cost}, \text{length}, \text{maximum} \rangle$

- $[\![\texttt{nil}]\!] = \langle 0, 0, 0 \rangle$
- $[\![\texttt{cons}(x, a)]\!] = \langle x_{\text{cost}} + a_{\text{cost}}, a_{\text{len}} + 1, \max(x_{\text{size}}, a_{\max}) \rangle$
- $[\![\texttt{map}(F, a)]\!] = \langle \text{cost}, \text{length}, \text{maximum} \rangle$, where:
  - length: a_{len}
  - maximum: $F(\ \langle a_{\text{cost}}, a_{\max} \rangle\ )_s$
  - cost: $(a_{\text{len}} + 1) * (F(\ \langle a_{\text{cost}}, a_{\max} \rangle\ )_{\text{cost}} + 1)$

Higher-order tuple interpretations: an example

$$\begin{array}{rcl} \texttt{nil} & :: & \mathsf{list} \\ \texttt{cons} & :: & [\mathsf{nat} \times \mathsf{list}] \Rightarrow \mathsf{list} \\ \texttt{map} & :: & [(\mathsf{nat} \Rightarrow \mathsf{nat}) \times \mathsf{list}] \Rightarrow \mathsf{list} \end{array}$$

$$\begin{array}{rcl} \texttt{map}(F, \texttt{nil}) & \to & \texttt{nil} \\ \texttt{map}(F, \texttt{cons}(x, a)) & \to & \texttt{cons}(F \cdot x, \texttt{map}(F, a)) \end{array}$$

Semantics: $(\!|\mathsf{list}|\!) = \langle \mathsf{cost}, \mathsf{length}, \mathsf{maximum} \rangle$

- $[\![\texttt{nil}]\!] = \langle 0, 0, 0 \rangle$
- $[\![\texttt{cons}(x, a)]\!] = \langle x_{\mathsf{cost}} + a_{\mathsf{cost}}, a_{\mathsf{len}} + 1, \max(x_{\mathsf{size}}, a_{\mathsf{max}}) \rangle$
- $[\![\texttt{map}(F, a)]\!] = \langle \mathsf{cost}, \mathsf{length}, \mathsf{maximum} \rangle$, where:
  - length: a_{len}
  - maximum: $F(\ \langle a_{\mathsf{cost}}, a_{\mathsf{max}} \rangle\ )_s$
  - cost: $(a_{\mathsf{len}} + 1) * (F(\ \langle a_{\mathsf{cost}}, a_{\mathsf{max}} \rangle\ )_{\mathsf{cost}} + 1)$

Roughly: $[\![\texttt{map}]\!](F, \langle \mathsf{cost}, \mathsf{len}, \mathsf{max} \rangle)_{\mathsf{cost}} \approx \mathsf{len} * F(\langle \mathsf{cost}, \mathsf{max} \rangle)_{\mathsf{cost}}$

Outline

Higher-order Feasibility

HO Rewriting and Tuple Interpretations

Runtime Complexity

BFFs Characterization

Recall: runtime complexity

Runtime complexity:

$n \mapsto$ "maximum derivation height for a basic term of size n"

Basic term: $\mathtt{function}(\mathtt{data}, \ldots, \mathtt{data})$

Example: $\mathtt{mul}(\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{0}))))), \mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{0}))))))))$

Recall: runtime complexity

Runtime complexity:
$n \mapsto$ "maximum derivation height for a basic term of size n"
Basic term: $\mathtt{function}(\mathtt{data}, \ldots, \mathtt{data})$
Example: $\mathtt{mul}(\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{0}))))), \mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{0}))))))))$

Problem: does this make sense for higher-order rewriting?

Higher-order runtime complexity?

Runtime complexity:
$n \mapsto$ "maximum derivation height for a basic term of size n"
Basic term: $\texttt{function}(\texttt{data}, \ldots, \texttt{data})$

Higher-order runtime complexity?

Runtime complexity:
$n \mapsto$ "maximum derivation height for a basic term of size n"
Basic term: $\mathtt{function}(\mathtt{data}, \ldots, \mathtt{data})$

- $\mathtt{map}(\lambda x.\mathtt{s}(x), \text{some lst})$?

Higher-order runtime complexity?

Runtime complexity:
$n \mapsto$ "maximum derivation height for a basic term of size n"
Basic term: $\texttt{function}(\texttt{data}, \ldots, \texttt{data})$

- $\texttt{map}(\lambda x.\texttt{s}(x), \text{some lst})$?
- $\texttt{f}(\lambda x.\texttt{cons}(x, \texttt{cons}(x, \texttt{nil})), \text{some data})$?

Higher-order runtime complexity?

Runtime complexity:
$n \mapsto$ "maximum derivation height for a basic term of size n"
Basic term: $\mathtt{function}(\mathtt{data}, \ldots, \mathtt{data})$

- $\mathtt{map}(\lambda x.\mathtt{s}(x), \text{some lst})$?
- $\mathtt{f}(\lambda x.\mathtt{cons}(x, \mathtt{cons}(x, \mathtt{nil})), \text{some data})$?

Choice: data must be a first-order `constructor` term.

Higher-order runtime complexity examples

$$\begin{array}{rcl} \mathtt{add}(\mathtt{0}, y) & \to & y \\ \mathtt{add}(\mathtt{s}(x), y) & \to & \mathtt{add}(x, \mathtt{s}(y)) \\ \mathtt{map}(F, \mathtt{nil}) & \to & \mathtt{nil} \\ \mathtt{map}(F, \mathtt{cons}(x, a)) & \to & \mathtt{cons}(F \cdot x, \mathtt{map}(F, a)) \end{array}$$

Terms of interest: $\mathtt{map}(\lambda y.\mathtt{add}(s, y), t)$

Higher-order runtime complexity examples

$$
\begin{array}{rcl}
\mathtt{add}(\mathtt{0}, y) & \rightarrow & y \\
\mathtt{add}(\mathtt{s}(x), y) & \rightarrow & \mathtt{add}(x, \mathtt{s}(y)) \\
\mathtt{map}(F, \mathtt{nil}) & \rightarrow & \mathtt{nil} \\
\mathtt{map}(F, \mathtt{cons}(x, a)) & \rightarrow & \mathtt{cons}(F \cdot x, \mathtt{map}(F, a)) \\
\mathtt{start}(x, a) & \rightarrow & \mathtt{map}(\lambda y.\mathtt{add}(x, y), a)
\end{array}
$$

Terms of interest: $\mathtt{map}(\lambda y.\mathtt{add}(s, y), t)$

Higher-order runtime complexity examples

$$
\begin{array}{rcl}
\mathtt{add}(\mathtt{0}, y) & \to & y \\
\mathtt{add}(\mathtt{s}(x), y) & \to & \mathtt{add}(x, \mathtt{s}(y)) \\
\mathtt{map}(F, \mathtt{nil}) & \to & \mathtt{nil} \\
\mathtt{map}(F, \mathtt{cons}(x, a)) & \to & \mathtt{cons}(F \cdot x, \mathtt{map}(F, a)) \\
\mathtt{start}(x, a) & \to & \mathtt{map}(\lambda y.\mathtt{add}(x, y), a)
\end{array}
$$

Terms of interest: $\mathtt{map}(\lambda y.\mathtt{add}(s, y), t)$

Basic term: $\mathtt{start}(\mathtt{s}^n(\mathtt{0}), \mathtt{cons}(\mathtt{s}^a(\mathtt{0}), \mathtt{cons}(\mathtt{s}^b(\mathtt{0}), \ldots, \mathtt{nil})))$

Higher-order runtime complexity examples

$$
\begin{array}{rcl}
\mathtt{add}(\mathtt{0}, y) & \to & y \\
\mathtt{add}(\mathtt{s}(x), y) & \to & \mathtt{add}(x, \mathtt{s}(y)) \\
\mathtt{map}(F, \mathtt{nil}) & \to & \mathtt{nil} \\
\mathtt{map}(F, \mathtt{cons}(x, a)) & \to & \mathtt{cons}(F \cdot x, \mathtt{map}(F, a)) \\
\mathtt{start}(x, a) & \to & \mathtt{map}(\lambda y.\mathtt{add}(x, y), a)
\end{array}
$$

Terms of interest: $\mathtt{map}(\lambda y.\mathtt{add}(s, y), t)$

Basic term: $\mathtt{start}(\mathtt{s}^n(\mathtt{0}), \mathtt{cons}(\mathtt{s}^a(\mathtt{0}), \mathtt{cons}(\mathtt{s}^b(\mathtt{0}), \dots, \mathtt{nil})))$

Runtime complexity: $n \mapsto \mathcal{O}(n^2)$

Higher-order runtime complexity examples

$$
\begin{array}{rcl}
\mathtt{add}(\mathtt{0}, y) & \to & y \\
\mathtt{add}(\mathtt{s}(x), y) & \to & \mathtt{add}(x, \mathtt{s}(y)) \\
\mathtt{map}(F, \mathtt{nil}) & \to & \mathtt{nil} \\
\mathtt{map}(F, \mathtt{cons}(x, a)) & \to & \mathtt{cons}(F \cdot x, \mathtt{map}(F, a)) \\
\mathtt{start}(x, a) & \to & \mathtt{map}(\lambda y.\mathtt{add}(x, y), a)
\end{array}
$$

Terms of interest: $\mathtt{map}(\lambda y.\mathtt{add}(s, y), t)$

Basic term: $\mathtt{start}(\mathtt{s}^n(\mathtt{0}), \mathtt{cons}(\mathtt{s}^a(\mathtt{0}), \mathtt{cons}(\mathtt{s}^b(\mathtt{0}), \dots, \mathtt{nil})))$

Runtime complexity: $n \mapsto \mathcal{O}(n^2)$ (length of t * size of s)

Heretofore...

- A simple idea: algebra interpretations with set $= \mathbb{N}^p$,

Heretofore...

- A simple idea: algebra interpretations with set $= \mathbb{N}^p$,
- important usage: different sets for different sorts,

Heretofore...

- A simple idea: algebra interpretations with set $= \mathbb{N}^p$,
- important usage: different sets for different sorts,
- essentially: a generalization of matrix interpretations,

Heretofore...

- A simple idea: algebra interpretations with set $= \mathbb{N}^p$,
- important usage: different sets for different sorts,
- essentially: a generalization of matrix interpretations,
- runtime complexity still makes higher-order sense (somewhat)

Heretofore...

- A simple idea: algebra interpretations with set $= \mathbb{N}^p$,
- important usage: different sets for different sorts,
- essentially: a generalization of matrix interpretations,
- runtime complexity still makes higher-order sense (somewhat)
- a more expressive complexity notion?

Outline

Higher-order Feasibility

HO Rewriting and Tuple Interpretations

Runtime Complexity

BFFs Characterization

How to characterize BFFs by Rewriting?

In order to capture BFFs we need to:

- show that every TRS **satisfying certain conditions** represent a BFF

How to characterize BFFs by Rewriting?

In order to capture BFFs we need to:

- show that every TRS **satisfying certain conditions** represent a BFF
- show that every BFF can be embedded as a TRS

How to characterize BFFs by Rewriting?

In order to capture BFFs we need to:

- show that every TRS **satisfying certain conditions** **represent** a BFF

How to characterize BFFs by Rewriting?

In order to capture BFFs we need to:

- show that every TRS **satisfying certain conditions** **represent** a BFF
  - we limit constructor symbols to **additive interpretations**

How to characterize BFFs by Rewriting?

In order to capture BFFs we need to:

- show that every TRS **satisfying certain conditions** **represent** a BFF
  - we limit constructor symbols to **additive interpretations**
  - all defined symbols have polynomial bounded interpretations

How to characterize BFFs by Rewriting?

In order to capture BFFs we need to:

- show that every TRS **satisfying certain conditions** **represent** a BFF
  - we limit constructor symbols to **additive interpretations**
  - all defined symbols have polynomial bounded interpretations
  - we add an infinite number of extra function symbols `f` to represent the **calls to ORACLES**

How to characterize BFFs by Rewriting?

In order to capture BFFs we need to:

- show that every TRS **satisfying certain conditions** **represent** a BFF
  - we limit constructor symbols to **additive interpretations**
  - all defined symbols have polynomial bounded interpretations
  - we add an infinite number of extra function symbols `f` to represent the **calls to ORACLES**
  - the cost int. of each oracle call is 1 and the size is polynomially bounded

How to characterize BFFs by Rewriting?

In order to capture BFFs we need to:

- show that every TRS **satisfying certain conditions** represent a BFF
- show that every BFF can be embedded as a TRS
  - $\mathtt{BLP}_2$ is a second order imperative stateful programming language

How to characterize BFFs by Rewriting?

In order to capture BFFs we need to:

- show that every TRS **satisfying certain conditions** represent a BFF
- show that every BFF can be embedded as a TRS
  - $\mathtt{BLP}_2$ is a second order imperative stateful programming language
  - a functional is in BFF iff it can be computed by a $\mathtt{BLP}_2$ program

How to characterize BFFs by Rewriting?

In order to capture BFFs we need to:

- show that every TRS **satisfying certain conditions** represent a BFF
- show that every BFF can be embedded as a TRS
  - $\mathtt{BLP}_2$ is a second order imperative stateful programming language
  - a functional is in BFF iff it can be computed by a $\mathtt{BLP}_2$ program
  - we then show that all $\mathtt{BLP}_2$ programs can be computed by second order TRSs with polynomial interpretations

Overview

- tuple interpretations allow us to split computation information into different cost and size components

Overview

- tuple interpretations allow us to split computation information into different cost and size components
- this ability allowed us to properly model oracle calls and bound their costs

Overview

- tuple interpretations allow us to split computation information into different cost and size components
- this ability allowed us to properly model oracle calls and bound their costs
- it adds expressivity to the complexity measure

Overview

- tuple interpretations allow us to split computation information into different cost and size components
- this ability allowed us to properly model oracle calls and bound their costs
- it adds expressivity to the complexity measure
- we can **implicitly** capture higher-order Feasibility!

Overview

- tuple interpretations allow us to split computation information into different cost and size components
- this ability allowed us to properly model oracle calls and bound their costs
- it adds expressivity to the complexity measure
- we can **implicitly** capture higher-order Feasibility!
- it is very interesting!

Overview

- tuple interpretations allow us to split computation information into different cost and size components
- this ability allowed us to properly model oracle calls and bound their costs
- it adds expressivity to the complexity measure
- we can **implicitly** capture higher-order Feasibility!
- it is very interesting!

Overview

- tuple interpretations allow us to split computation information into different cost and size components
- this ability allowed us to properly model oracle calls and bound their costs
- it adds expressivity to the complexity measure
- we can **implicitly** capture higher-order Feasibility!
- it is very interesting!

Thank you!

BFFs extra definitions

Definition

Given a functional F we say that

- F is defined from $H, G_1, \ldots, G_l$ by functional composition if for all $\vec{f}$ and $\vec{x}$,
$$F(\vec{f}, \vec{x}) = H(\vec{f}, G_1(\vec{f}, \vec{x}), \ldots, G_l(\vec{f}, \vec{x})).$$
- F is defined from G by expansion if for all $\vec{f}$, $\vec{g}$, $\vec{x}$, and $\vec{y}$,
$$F(\vec{f}, \vec{g}, \vec{x}, \vec{y}) = G(\vec{f}, \vec{x}).$$

More than matrix and polynomial interpretations!

$$
\begin{aligned}
\mathtt{minus}(x, \mathtt{0}) &\to x \\
\mathtt{minus}(\mathtt{s}(x), \mathtt{s}(y)) &\to \mathtt{minus}(x, y) \\
\mathtt{quot}(\mathtt{0}, \mathtt{s}(y)) &\to \mathtt{0} \\
\mathtt{quot}(\mathtt{s}(x), \mathtt{s}(y)) &\to \mathtt{s}(\mathtt{quot}(\mathtt{minus}(x, y), \mathtt{s}(y)))
\end{aligned}
$$

More than matrix and polynomial interpretations!

$$\begin{array}{rcl} \mathtt{minus}(x, \mathtt{0}) & \to & x \\ \mathtt{minus}(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{minus}(x, y) \\ \mathtt{quot}(\mathtt{0}, \mathtt{s}(y)) & \to & \mathtt{0} \\ \mathtt{quot}(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{s}(\mathtt{quot}(\mathtt{minus}(x, y), \mathtt{s}(y))) \end{array}$$

- Cannot be done with polynomial interpretations, since always $[\![\mathtt{minus}(x, y)]\!] \geq [\![y]\!]$.

More than matrix and polynomial interpretations!

$$\begin{array}{rcl}
\mathtt{minus}(x, \mathtt{0}) & \rightarrow & x \\
\mathtt{minus}(\mathtt{s}(x), \mathtt{s}(y)) & \rightarrow & \mathtt{minus}(x, y) \\
\mathtt{quot}(\mathtt{0}, \mathtt{s}(y)) & \rightarrow & \mathtt{0} \\
\mathtt{quot}(\mathtt{s}(x), \mathtt{s}(y)) & \rightarrow & \mathtt{s}(\mathtt{quot}(\mathtt{minus}(x, y), \mathtt{s}(y)))
\end{array}$$

- Cannot be done with polynomial interpretations, since always $[\![\mathtt{minus}(x, y)]\!] \geq [\![y]\!]$.
- Cannot be done with matrix interpretations due to duplication of y.

More than matrix and polynomial interpretations!

$$
\begin{array}{rcl}
\mathtt{minus}(x, \mathtt{0}) & \to & x \\
\mathtt{minus}(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{minus}(x, y) \\
\mathtt{quot}(\mathtt{0}, \mathtt{s}(y)) & \to & \mathtt{0} \\
\mathtt{quot}(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{s}(\mathtt{quot}(\mathtt{minus}(x, y), \mathtt{s}(y)))
\end{array}
$$

- Cannot be done with polynomial interpretations, since always $[\![\mathtt{minus}(x, y)]\!] \geq [\![y]\!]$.
- Cannot be done with matrix interpretations due to duplication of y.
- Can be done with tuple interpretations!

$$
\begin{array}{rcl}
[\![\mathtt{0}]\!] & = & \langle 0, 0 \rangle \\
[\![\mathtt{s}(x)]\!] & = & \langle x_{\text{cost}}, x_{\text{size}} + 1 \rangle \\
[\![\mathtt{minus}(x, y)]\!] & = & \langle x_{\text{cost}} + y_{\text{cost}} + y_{\text{size}} + 1, x_{\text{size}} \rangle \\
[\![\mathtt{quot}(x, y)]\!] & = & \langle x_{\text{cost}} + y_{\text{cost}} + x_{\text{size}} + x_{\text{size}} * (y_{\text{size}} + y_{\text{cost}}) + 1, \\
 & & x_{\text{size}} \rangle
\end{array}
$$

Some other examples

$$
\begin{array}{rcl}
\mathtt{filter}(F, \mathtt{nil}) & \to & \mathtt{nil} \\
\mathtt{filter}(F, \mathtt{cons}(x, a)) & \to & \mathtt{consif}(F \cdot x, x, \mathtt{filter}(F, a)) \\
\mathtt{consif}(\mathtt{true}, x, a) & \to & \mathtt{cons}(x, a) \\
\mathtt{consif}(\mathtt{false}, x, a) & \to & a
\end{array}
$$

Cost: $1 + (a_{\mathsf{len}} + 1) * (2 + a_{\mathsf{cost}} + F(\langle a_{\mathsf{cost}}, a_{\mathsf{max}} \rangle)_{\mathsf{cost}})$

Some other examples

$$\begin{array}{rcl}
\mathtt{filter}(F, \mathtt{nil}) & \to & \mathtt{nil} \\
\mathtt{filter}(F, \mathtt{cons}(x, a)) & \to & \mathtt{consif}(F \cdot x, x, \mathtt{filter}(F, a)) \\
\mathtt{consif}(\mathtt{true}, x, a) & \to & \mathtt{cons}(x, a) \\
\mathtt{consif}(\mathtt{false}, x, a) & \to & a
\end{array}$$

Cost: $1 + (a_{\mathsf{len}} + 1) * (2 + a_{\mathsf{cost}} + F(\langle a_{\mathsf{cost}}, a_{\mathsf{max}} \rangle)_{\mathsf{cost}})$

Roughly:
$[\![\mathtt{filter}]\!](F, \langle \mathsf{cost}, \mathsf{len}, \mathsf{max} \rangle)_{\mathsf{cost}} \approx \mathsf{len} * F(\langle \mathsf{cost}, \mathsf{max} \rangle)_{\mathsf{cost}} + \mathsf{len} * \mathsf{cost}$

Some other examples

$$\begin{array}{rcl} \mathtt{filter}(F, \mathtt{nil}) & \to & \mathtt{nil} \\ \mathtt{filter}(F, \mathtt{cons}(x, a)) & \to & \mathtt{consif}(F \cdot x, x, \mathtt{filter}(F, a)) \\ \mathtt{consif}(\mathtt{true}, x, a) & \to & \mathtt{cons}(x, a) \\ \mathtt{consif}(\mathtt{false}, x, a) & \to & a \end{array}$$

Cost: $1 + (a_{\mathsf{len}} + 1) * (2 + a_{\mathsf{cost}} + F(\langle a_{\mathsf{cost}}, a_{\mathsf{max}} \rangle)_{\mathsf{cost}})$

Roughly:

$$[\![\mathtt{filter}]\!](F, \langle \mathsf{cost}, \mathsf{len}, \mathsf{max} \rangle)_{\mathsf{cost}} \approx \underbrace{\mathsf{len} * F(\langle \mathsf{cost}, \mathsf{max} \rangle)_{\mathsf{cost}}}_{\text{map-like component!}} + \mathsf{len} * \mathsf{cost}$$

Some other examples

$$\begin{array}{rcl} \mathtt{rec}(\mathtt{0}, y, F) & \to & y \\ \mathtt{rec}(\mathtt{s}(x), y, F) & \to & F \cdot x \cdot \mathtt{rec}(x, y, F) \end{array}$$

Cost: $\mathit{Helper}[x, F]^{x_{\mathsf{len}}+1}(\langle 1 + y_{\mathsf{cost}}, y_{\mathsf{size}} \rangle)$ where
$\mathit{Helper}[x, F] = z \mapsto \langle\ F(x, z)_{\mathsf{cost}},\ \max(z_{\mathsf{size}}, F(x, z)_{\mathsf{size}})\ \rangle$

Some other examples

$$\begin{array}{rcl} \mathtt{rec}(\mathtt{0}, y, F) & \rightarrow & y \\ \mathtt{rec}(\mathtt{s}(x), y, F) & \rightarrow & F \cdot x \cdot \mathtt{rec}(x, y, F) \end{array}$$

Cost: $\mathit{Helper}[x, F]^{x_{\mathsf{len}}+1}(\langle 1 + y_{\mathsf{cost}}, y_{\mathsf{size}} \rangle)$ where
$\mathit{Helper}[x, F] = z \mapsto \langle\ F(x, z)_{\mathsf{cost}},\ \max(z_{\mathsf{size}}, F(x, z)_{\mathsf{size}})\ \rangle$

Roughly: $[\![\mathtt{rec}]\!](\langle \mathsf{cost}, \mathsf{size} \rangle, y, F) \approx$
$(z \mapsto F(\langle \mathsf{cost}, \mathsf{size} \rangle, z))^{\mathsf{size}}(x)$.

Some other examples

$$\begin{array}{rcl} \mathtt{rec}(0, y, F) & \to & y \\ \mathtt{rec}(\mathtt{s}(x), y, F) & \to & F \cdot x \cdot \mathtt{rec}(x, y, F) \end{array}$$

Cost: $\mathit{Helper}[x, F]^{x_{\mathsf{len}}+1}(\langle 1 + y_{\mathsf{cost}}, y_{\mathsf{size}} \rangle)$ where
$\mathit{Helper}[x, F] = z \mapsto \langle\ F(x, z)_{\mathsf{cost}},\ \max(z_{\mathsf{size}}, F(x, z)_{\mathsf{size}})\ \rangle$

Roughly: $[\![\mathtt{rec}]\!](\langle \mathsf{cost}, \mathsf{size} \rangle, y, F) \approx$
$(z \mapsto F(\langle \mathsf{cost}, \mathsf{size} \rangle, z))^{\mathsf{size}}(x)$.

Compare: $[\![\mathtt{fold}]\!](F, x, \langle \mathsf{cost}, \mathsf{len}, \max \rangle) \approx$
$(z \mapsto F(z, \langle \mathsf{cost}, \max \rangle))^{\mathsf{len}}(x)$.

Some other examples

$$
\begin{array}{rcl}
\mathtt{der}(\lambda x.y) & \rightarrow & \lambda z.\mathtt{0} \\
\mathtt{der}(\lambda x.x) & \rightarrow & \lambda z.\mathtt{one} \\
\mathtt{der}(\lambda x.\mathtt{sin}(x)) & \rightarrow & \lambda z.\mathtt{cos}(z) \\
\mathtt{der}(\lambda x.\mathtt{cos}(x)) & \rightarrow & \lambda z.\mathtt{minus}(\mathtt{sin}(z)) \\
\mathtt{der}(\lambda x.\mathtt{plus}(F \cdot x, G \cdot x)) & \rightarrow & \lambda z.\mathtt{plus}(\mathtt{der}(F) \cdot z, \mathtt{der}(G) \cdot z) \\
\mathtt{der}(\lambda x.\mathtt{times}(F \cdot x, G \cdot x)) & \rightarrow & \lambda z.\mathtt{plus}(\mathtt{times}(\mathtt{der}(F) \cdot z, G \cdot z), \\
 & & \qquad \mathtt{times}(F \cdot z, \mathtt{der}(G) \cdot z)) \\
\mathtt{der}(\lambda x.\mathtt{ln}(F \cdot x)) & \rightarrow & \lambda z.\mathtt{div}(\mathtt{der}(F) \cdot z, F \cdot z)
\end{array}
$$

Cost of $\mathtt{der}(F, z)$***:*** $1 + F(z)_{\mathsf{cost}} + 2 * F(z)_{\mathsf{size}} + F(z)_{ndif} * F(z)_{\mathsf{cost}}$
$\approx F(z)_{ndif} * F(z)_{\mathsf{cost}}$

Thank you!