On Quasi-Interpretations, Blind Abstractions and Implicit Complexity

Patrick Baillot Ugo Dal Lago Jean-Yves Moyen

LIPN Paris 13, Univ. di Bologna/Paris 7 LIPN Paris 13

journées GdT LAC, Chambéry, 9/2/2007

(projet ANR NOCoST)

From termination to complexity?

- termination is widely studied: polynomial interpretations, RPO, . . .
- however in practice computational complexity often relevant (eg for feasible termination)
  $\Rightarrow$ how to guarantee/verify that a program is polynomial time (Ptime) ?
- *Implicit computational complexity (ICC)*: study calculi with intrinsic complexity properties (e.g. Ptime) ,
  - primitive recursive programs
  - typed lambda-calculi, linear logic
  - TRS

  Leivant, Jones, Girard . . .

Implicit Computational Complexity and intensional expressivity

- Typical ICC results:
  any *program* of the class $\mathcal{C}$ computes a Ptime function,
  and any Ptime *function* can be computed by at least one program of $\mathcal{C}$.
  (using a simulation of Ptime Turing machines)
- *Intensional expressivity*: which algorithmic patterns are available in an ICC system ?
  TRS: advantage of general recursion, pattern-matching
- *Term rewriting (TRS) and Quasi-interpretations* (Bonfante - Marion -Moyen):
  an easy-to-use and quite general ICC system

  idea: combine 2 ingredients

  RPO + *size argument* (quasi-interpretation) how to study the intensional expressivity of ICC calculi?

- examples
- we propose to consider program transformations or abstractions, to find out necessary conditions on programs

here we study the Quasi-interpretations (QI) approach (P-criterion of Marion *et al.*), and define for that *blind abstraction* of programs.

this way we provide a *necessary condition* on programs meeting the P-criterion on QI.

Applications:

- property of Bellantoni-Cook programs,
- extensions of the P-criterion.

Outline

1. Background: TRS and Quasi-interpretations.
   P-criterion ([BMM06]).
2. Blind abstractions.
   Blindly polytime programs.
3. P-criterion and blind abstractions.
   application: Bellantoni-Cook; extensions of P-criterion.

< > - +

1. Programs as TRS

Definition 1 (Syntax) *Terms and equations are defined by:*

$$
\begin{array}{lrcl}
(values) & \mathcal{T}(\mathcal{C}) \ni v & ::= & \mathbf{c}(v_1, \cdots, v_n) \\
(terms) & \mathcal{T}(\mathcal{C}, \mathcal{F}, \mathcal{X}) \ni t & ::= & x \mid \mathbf{c}(t_1, \cdots, t_n) \mid \mathtt{f}(t_1, \cdots, t_n) \\
(patterns) & \mathcal{P} \ni p & ::= & x \mid \mathbf{c}(p_1, \cdots, p_n) \\
(equations) & \mathcal{D} \ni d & ::= & \mathtt{f}(p_1, \cdots, p_n) \to t
\end{array}
$$

where $x \in \mathcal{X}$, $\mathtt{f} \in \mathcal{F}$, *and* $\mathbf{c} \in \mathcal{C}$.

A program: $\mathtt{f} = \langle \mathcal{X}, \mathcal{C}, \mathcal{F}, \mathcal{E} \rangle$ where $\mathcal{E}$ is a set of equations in $\mathcal{D}$.
In equation $\mathtt{f}(\vec{p}) \to t$, each variable in t also appears in $\mathtt{f}(\vec{p})$.
programs are constructor (term-rewriting) systems.

in general, we don't require determinism condition

< > - +

Operational semantics

- Call-by-value semantics:

$$\frac{\mathbf{c} \in \mathcal{C} \quad t_i \downarrow v_i}{\mathbf{c}(t_1, \cdots, t_n) \downarrow \mathbf{c}(v_1, \cdots, v_n)} \text{ (Constructor)}$$

$$\frac{\exists j, t_j \notin \mathcal{T}(\mathcal{C}) \quad t_i \downarrow v_i \quad \mathtt{f}(v_1, \cdots, v_n) \downarrow v}{\mathtt{f}(t_1, \cdots, t_n) \downarrow v} \text{ (Split)}$$

$$\frac{\mathtt{f}(p_1, \cdots, p_n) \to r \in \mathcal{E} \quad \sigma \in \mathfrak{S} \quad p_i\sigma = v_i \quad r\sigma \downarrow v}{\mathtt{f}(v_1, \cdots, v_n) \downarrow v} \text{ (Function)}$$

- Call-by-value semantics with cache:
  - corresponds to programming with memoisation: avoid recomputing of values,,
  - judgements $\langle C, t \rangle$, where C is a *cache*,
  - $(update)$ and $(read)$ rules.

< > - +

Example of program

$$
\begin{array}{lll}
\mathtt{f}(\mathbf{s}_0\mathbf{s}_i x) & \rightarrow \mathtt{append}(\mathtt{f}(\mathbf{s}_1 x), \mathtt{f}(\mathbf{s}_1 x)) & i = 0, 1 \\
\mathtt{f}(\mathbf{s}_1 x) & \rightarrow x & \\
\mathtt{f}(\mathbf{nil}) & \rightarrow \mathbf{nil} & \\
\mathtt{append}(\mathbf{s}_i x, y) & \rightarrow \mathbf{s}_i \mathtt{append}(x, y) & \\
\mathtt{append}(\mathbf{nil}, y) & \rightarrow y &
\end{array}
$$

π is the following derivation:

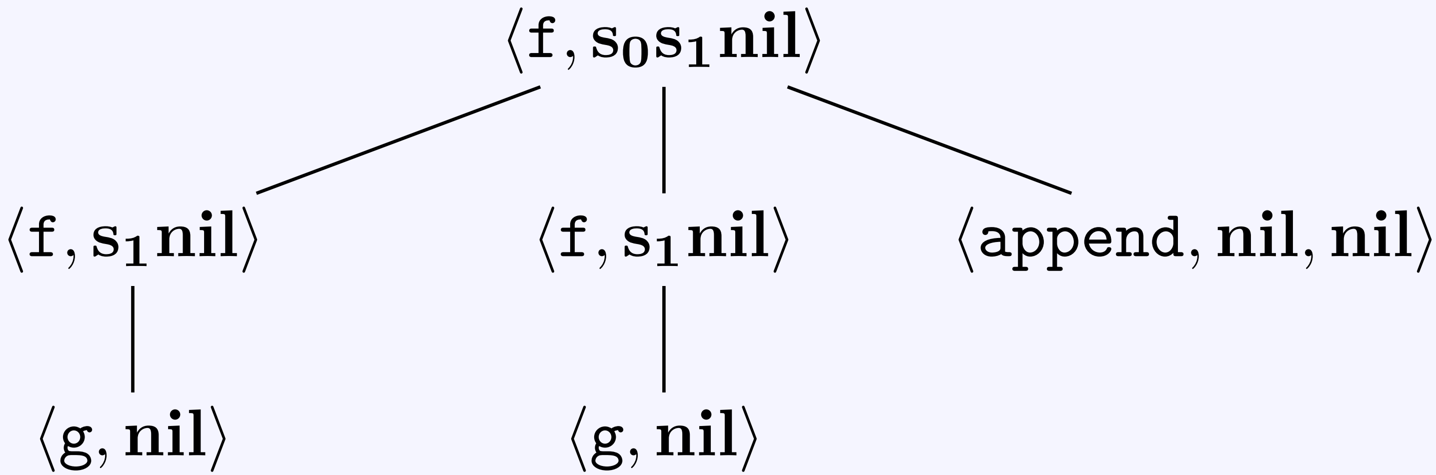
$$
\dfrac{\dfrac{\mathbf{nil} \downarrow \mathbf{nil}}{\mathtt{f}(\mathbf{s}_1\mathbf{nil}) \downarrow \mathbf{nil}} \quad \dfrac{\mathbf{nil} \downarrow \mathbf{nil}}{\mathtt{f}(\mathbf{s}_1\mathbf{nil}) \downarrow \mathbf{nil}} \quad \dfrac{\mathbf{nil} \downarrow \mathbf{nil}}{\mathtt{append}(\mathbf{nil}, \mathbf{nil}) \downarrow \mathbf{nil}}}{\dfrac{\mathtt{append}(\mathtt{f}(\mathbf{s}_1\mathbf{nil}), \mathtt{f}(\mathbf{s}_1\mathbf{nil})) \downarrow \mathbf{nil}}{\mathtt{f}(\mathbf{s}_0\mathbf{s}_1\mathbf{nil}) \downarrow \mathbf{nil}}}
$$

< > - +

Call trees

Call trees are a tool to analyse the execution of programs.

- Let $\pi : t \downarrow v$ be a reduction proof. Its *call trees* is the set of trees Θ_π obtained by only keeping terms $f(\vec{w})$ in conclusions of (Function) rules.
  In our example program, the following is a call-tree for $\langle \mathtt{f}, \mathbf{s}_0\mathbf{s}_1\mathbf{nil}\rangle$:



- for the *Call-by-value semantics with cache*: also a notion of *Call dag*, obtained by identifying some nodes in the call tree.

< > - +

Termination orderings

- *precedence* $\preceq_{\mathcal{F}}$: preorder over $\mathcal{F} \bigcup \mathcal{C}$.
  $\approx_{\mathcal{F}}$ associated equivalence relation.
- *separating* precedence: constructors $\preceq_{\mathcal{F}}$ functions
- *fair* precedence: for constructors $\mathbf{c}, \mathbf{d}$ with same arity, $\mathbf{c} \approx_{\mathcal{F}} \mathbf{d}$,
  strict precedence: distinct constructors not comparable for $\preceq_{\mathcal{F}}$.
- *product extension* of an ordering $\preceq$: extension over tuples such that $(m_1, \cdots, m_k) \prec^p (n_1, \cdots, n_k)$ iff
  (i) $\forall i, m_i \preceq n_i$ and (ii) $\exists j$ such that $m_j \prec n_j$.

< > - +

Termination orderings (continued)

$$\frac{s = t_i \text{ or } s \prec_{rpo} t_i}{s \prec_{rpo} \mathtt{f}(\dots, t_i, \dots)} \; \mathtt{f} \in \mathcal{F} \bigcup \mathcal{C} \qquad \frac{\forall i \; s_i \prec_{rpo} \mathtt{f}(t_1, \cdots, t_n) \qquad \mathtt{g} \prec_{\mathcal{F}} \mathtt{f}}{g(s_1, \cdots, s_m) \prec_{rpo} \mathtt{f}(t_1, \cdots, t_n)} \; \mathtt{f}, \mathtt{g} \in \mathcal{F} \bigcup \mathcal{C}$$

$$\frac{(s_1, \cdots, s_n) \prec^{p}_{rpo} (t_1, \cdots, t_n) \qquad \mathtt{f} \approx_{\mathcal{F}} \mathtt{g} \qquad \forall i \; s_i \prec_{rpo} \mathtt{f}(t_1, \cdots, t_n)}{\mathtt{g}(s_1, \cdots, s_n) \prec_{rpo} \mathtt{f}(t_1, \cdots, t_n)} \; \mathtt{f}, \mathtt{g} \in \mathcal{F} \bigcup \mathcal{C}$$

PPO: recursive path ordering $\prec_{rpo}$ obtained with separating precedence

EPPO: recursive path ordering $\prec_{rpo}$ obtained with fair precedence

Quasi-interpretations

idea: provide upper bound on *size* of intermediate values during computation

Let $b \in \mathcal{F} \bigcup \mathcal{C}$ with arity n. Its *assignment* is a function $(\!|b|\!) : (\mathbb{R})^n \rightarrow \mathbb{R}$ such that:

(Subterm) $(\!|b|\!)(X_1, \cdots, X_n) \geq X_i$ for all $1 \leq i \leq n$.

(Weak Monotonicity) $(\!|b|\!)$ is increasing (not strictly) wrt each variable.

(Additivity) $(\!|\mathbf{c}|\!)(X_1, \cdots, X_n) \geq \sum_{i=1}^{n} X_i + a$ if $\mathbf{c} \in \mathcal{C}$ (where $a \geq 1$).

(Polynomial) $(\!|b|\!)$ is bounded by a polynomial.

Assignments $(\!|.|\!)$ are extended to terms canonically. If t has n variables, then $(\!|t|\!) : (\mathbb{R})^n \rightarrow \mathbb{R}$.
we denote by $\geq$ the extensional order on functions.
if t is a subterm of s, then $(\!|s|\!) \geq (\!|t|\!)$

< > - +

P-criterion

Definition 2 (Quasi-interpretation) *An assignment $(\!|.|\!)$ of a program is a quasi-interpretation (QI) if for each equation $l \to r$, $(\!|l|\!) \geq (\!|r|\!)$.*

Ex: $(\!|\mathbf{s}_0|\!)(X) = (\!|\mathbf{s}_1|\!)(X) = X + 1$, $(\!|\mathbf{append}|\!)(X, Y) = X + Y$.

For inference, QI can be searched in a given function algebra, *e.g.* MaxPoly.

Theorem 1 (P-criterion, Bonfante-Marion-Moyen) *The set of functions computable by programs that*
(i) terminate by PPO, and (ii) admit a QI,
is exactly FP.

To execute the program with a polynomial bound, one must use a call-by-value semantics with cache.

Ex: insertion sort, longest-common-subsequence.

2. Blind abstractions

Idea: study properties of the program by considering an abstraction.

blind abstraction: all constructors of same arity are collapsed into one, we associate to a program $\mathtt{f}$ another one $\overline{\mathtt{f}}$, which is not deterministic.

More precisely, target language:

- variables: $\overline{\mathcal{X}} = \mathcal{X}$,
- function symbols: $\overline{\mathcal{F}} = \{\overline{\mathtt{f}} \mathbin{/} \mathtt{f} \in \mathcal{F}\}$.
- constructor symbols: the map $\overline{(.)}$ on constructor symbols defined by: $\overline{\mathbf{c}} = \overline{\mathbf{d}}$ iff $\mathbf{c}$ and $\mathbf{d}$ have the same arity. Then $\overline{\mathcal{C}} = \{\overline{\mathbf{c}} \mathbin{/} \mathbf{c} \in \mathcal{C}\}$.

The *blinding map* is then the natural map $\mathcal{B} : \mathcal{T}(\mathcal{C}, \mathcal{F}, \mathcal{X}) \longrightarrow \mathcal{T}(\overline{\mathcal{C}}, \overline{\mathcal{F}}, \overline{\mathcal{X}})$.

Ex: binary lists built over $\{\mathbf{s}_0, \mathbf{s}_1, \mathbf{nil}\}$ mapped to tally integers, built from $\{\mathbf{s}, \mathbf{0}\}$, where $\overline{\mathbf{s}_0} = \overline{\mathbf{s}_1} = \mathbf{s}$ and $\overline{\mathbf{nil}} = \mathbf{0}$.

Blinding and complexity definitions

Definition 3 (Strongly polytime) *A (possibly) non-deterministic program* $\mathtt{f}$ *is strongly polytime if there exists a polynomial* $p_{\mathtt{f}} : \mathbb{N}^n \to \mathbb{N}$ *such that for every sequence* $v_1, \cdots, v_n$ *and any* $\pi : \mathtt{f}(v_1, \cdots, v_n) \downarrow u$*, it holds that* $|\pi| \leq p_{\mathtt{f}}(v_1, \ldots, v_n)$*.*

Definition 4 (Blindly polytime) *A program* $\mathtt{f}$ *is blindly polytime if its blind abstraction* $\overline{\mathtt{f}}$ *is strongly polytime.*

< > - +

Blindly polytime programs

Observe that:

Fact 1 *If a program* `f` *is blindly polytime, then it is polytime in the call-by-value semantics.*

For instance, the Quicksort algorithm is blindly polytime.

Remark:

say an *error* is replacement of a constructor by a constructor of same arity

blindly Ptime program= program remaining Ptime, no matter the number of errors occurring during execution.

Example

$\mathtt{f}$			$\overline{\mathtt{f}}$		
$\mathtt{f}(\mathbf{s}_0\mathbf{s}_i x)$	$\rightarrow$	$\mathtt{append}(\mathtt{f}(\mathbf{s}_1 x), \mathtt{f}(\mathbf{s}_1 x))$	$\overline{\mathtt{f}}(\mathbf{s}\mathbf{s}x)$	$\rightarrow$	$\overline{\mathtt{append}}(\overline{\mathtt{f}}(\mathbf{s}x), \overline{\mathtt{f}}(\mathbf{s}x))$
$\mathtt{f}(\mathbf{s}_1 x)$	$\rightarrow$	x	$\overline{\mathtt{f}}(\mathbf{s}x)$	$\rightarrow$	x
$\mathtt{f}(\mathbf{nil})$	$\rightarrow$	$\mathbf{nil}$	$\overline{\mathtt{f}}(0)$	$\rightarrow$	0
$\mathtt{append}(\mathbf{s}_i x, y)$	$\rightarrow$	$\mathbf{s}_i \mathtt{append}(x, y)$	$\overline{\mathtt{append}}(\mathbf{s}x, y)$	$\rightarrow$	$\mathbf{s}\ \overline{\mathtt{append}}(x, y)$
$\mathtt{append}(\mathbf{nil}, y)$	$\rightarrow$	y	$\overline{\mathtt{append}}(0, y)$	$\rightarrow$	y

$\mathtt{f}$ is Ptime but not blindly Ptime.

Indeed note $\underline{n} = \underbrace{\mathbf{s} \ldots \mathbf{s}}_{n}\ \mathbf{0}$, we have that $\overline{\mathtt{f}}(\underline{n})$ is reduced in an exponential number of steps, with a $\pi : \overline{\mathtt{f}}(\underline{n}) \downarrow \underline{2^{n-1}}$.

< > - +

Blinding and PPO, QI

- **Proposition 1** *The three following statements are equivalent:*
  (i) f terminates by EPPO,
  (ii) $\overline{f}$ terminates by EPPO,
  (iii) $\overline{f}$ terminates by PPO.

- An assignment for $\mathtt{f} = \langle \mathcal{X}, \mathcal{C}, \mathcal{F}, \mathcal{E} \rangle$ is *uniform* if all constructors of same arity have the same assignment.

  Proposition 2 *The program* $\mathtt{f}$ *admits a uniform quasi-interpretation iff* $\overline{\mathtt{f}}$ *admits a quasi-interpretation.*

< > - +

3. Linearity and the P-criterion

$\mathtt{f}$ RPO program and $\mathtt{g}$ a function in $\mathtt{f}$:

$\mathtt{g}$ is *linear* if in any equation $\mathtt{g}(\vec{p}) \rightarrow t$, there is at most one occurrence in t of a h such that $h \approx_{\mathcal{F}} \mathtt{g}$.

program $\mathtt{f}$ is linear if all its functions are linear.

Theorem 2 *Let* $\mathtt{f}$ *be a (possibly non deterministic) program which*

i) terminates by PPO,

ii) admits a quasi-interpretation,

iii) is linear.

Then $\mathtt{f}$ *is strongly polytime.*

< > - +

Blinding and the P-criterion

Theorem 3 *Let* `f` *be a (possibly non deterministic) program which*
i) terminates by PPO,
ii) admits a uniform quasi-interpretation,
iii) is linear.
Then `f` *is blindly polytime.*

Intuition: the criterion cannot perform analysis relying on case distinction on *content* of values.

< > - +

Blinding of Bellantoni-Cook programs

Bellantoni-Cook programs (BC): subclass of primitive recursive programs, defined by distinguishing *safe* and *normal* arguments:

$$f(x_1, \ldots, x_k; x_{k+1}, \ldots, x_n)$$

Each BC program can be turned into a linear program terminating by PPO.

Ex: Safe recursion construction:

(**Safe recursion**)

$$\mathtt{f}(\mathbf{0}, \vec{x}; \vec{y}) \rightarrow \mathtt{g}(\vec{x}; \vec{y})$$

$$\mathtt{f}(\mathbf{s}_i(z), \vec{x}; \vec{y}) \rightarrow \mathtt{h}_i(z, \vec{x}; \vec{y}, \mathtt{f}(z, \vec{x}; \vec{y})), i \in \{0, 1\}$$

with $\mathtt{g}, \mathtt{h}_i \in$ BC (previously defined)

one can build a uniform quasi-interpretation for each such program.

Theorem 4 *If* $\mathtt{f}$ *is a program of BC, then* $\mathtt{f}$ *is blindly polytime.*

Extension of the P-criterion

Ex. of program not terminating by PPO: "fast exponentiation algorithm" in base 4, that is using the recurrence $x^{4y} = ((x^y)^2)^2$

$$
\begin{array}{lll}
\mathtt{pow}(x, \mathtt{s}_0\mathtt{s}_0 y) & \to \mathtt{sq}(\mathtt{sq}(\mathtt{pow}(x, y))) & (x^{4y} = ((x^y)^2)^2) \\
\mathtt{pow}(x, \mathtt{s}_1\mathtt{s}_0 y) & \to \mathtt{mult}(x, \mathtt{pow}(x, \mathtt{s}_0\mathtt{s}_0 y)) & (x^{4y+1} = x^{4y} \times x) \\
\mathtt{pow}(x, \mathtt{s}_0\mathtt{s}_1 y) & \to \mathtt{sq}(\mathtt{mult}(x, \mathtt{pow}(x, \mathtt{s}_0 y))) & (x^{4y+2} = (x^{2y} \times x)^2) \\
\mathtt{pow}(x, \mathtt{s}_1\mathtt{s}_1 y) & \to \mathtt{mult}(x, \mathtt{sq}(\mathtt{mult}(x, \mathtt{pow}(x, \mathtt{s}_0 y)))) & (x^{4y+3} = (x^{2y} \times x)^2 \times x \\
\ldots & \to &
\end{array}
$$

but this program terminates by EPPO.

We extend to EPPO the P-criterion (for programs on lists over a finite alphabet):

Theorem 5 *The set of functions computed by programs over lists terminating by EPPO and admitting a QI is exactly* FP.

< > - +

An abstract P-criterion

Definition 5 (Bounded Values) *A program* $\mathtt{f} = \langle \mathcal{X}, \mathcal{C}, \mathcal{F}, \mathcal{E} \rangle$ *has polynomially bounded values iff for every* n*-ary function symbol* $\mathtt{g} \in \mathcal{F}$*, there is a polynomial* $p_{\mathtt{g}} : \mathbb{N} \to \mathbb{N}$ *such that for every state* η' *appearing in a call tree for* $\eta = (\mathtt{g}, v_1, \ldots, v_n)$*,* $|\eta'| \leq p_{\mathtt{g}}(|\eta|)$*.*

$\mathtt{f}$ admits a QI $\quad \Rightarrow \quad$ $\mathtt{f}$ has polynomially bounded values.

Theorem 6 *Let* f *be a deterministic program terminating by EPPO. Then the following two conditions are equivalent:*

1. f *has polynomially bounded values;*
2. f *is polytime in the call-by-value semantics with memoisation.*

< > - +

Conclusion and perspectives

- analyse ICC criteria by studying necessary conditions on programs:
  here we defined blind abstractions for TRS
- this is a possible way to understand the limitations of certain ICC criteria, and maybe to generalize them
- obtain necessary and sufficient conditions ?

Project:
NOCoST project (*New Tools for Complexity: Semantics and Types*): 2005-2008 (ANR).
Sites: LIPN Paris 13, PPS Paris 7.
http://www-lipn.univ-paris13.fr/nocost/