Quasi-interpretations

A way to control resources

Jean-Yves Marion

Ecole nationale supérieure des Mines de Nancy
Loria-INPL

February, 6th 2006

Outline

Shuffle

Example

$$\begin{aligned} \text{shuffle}(\epsilon, y) &\to y \\ \text{shuffle}(x, \epsilon) &\to x \\ \text{shuffle}(\mathbf{i}(x), \mathbf{j}(y)) &\to \mathbf{i}(\mathbf{j}(\text{shuffle}(x, y))) \qquad \mathbf{i}, \mathbf{j} \in \{\mathbf{0}, \mathbf{1}\} \end{aligned}$$

$$\begin{aligned} \text{shuffle}(\mathbf{10}(\epsilon), \mathbf{001}(\epsilon)) &\to \mathbf{10}(\text{shuffle}(\mathbf{0}(\epsilon)), \mathbf{01}(\epsilon)) \\ &\to \mathbf{1000}(\text{shuffle}(\epsilon, \mathbf{1}(\epsilon))) \\ &\to \mathbf{10001}(\epsilon) \end{aligned}$$

Domain of computations is the binary word struct. $\langle \epsilon, \mathbf{0}, \mathbf{1} \rangle$

Program Syntax

Constructor

$$v \quad ::= \quad \mathbf{c} \mid \mathbf{c}(v_1, \cdots, v_n)$$

$\mathbf{c} \in \mathcal{C}$ set of constructors

Patterns

$$p \quad ::= \quad \mathbf{c} \mid x \mid \mathbf{c}(p_1, \cdots, p_n)$$

x is a variable

Terms

$$t \quad ::= \quad \mathbf{c} \mid x \mid \mathbf{c}(t_1, \cdots, t_n) \mid \mathtt{f}(t_1, \cdots, t_n)$$

$\mathtt{f}$ is a fct symbol

Rules

$$d \quad ::= \quad \mathtt{f}(p_1, \cdots, p_n) \to t$$

A program $\mathtt{f}$ is a set of rewriting rules

Program rewriting

- A context is a special term $C[\_]$ with a hole
- $u \rightarrow v$ iff there is a substitution σ and $\mathtt{f}(p_1, \cdots, p_n) \rightarrow t$ such that $u = \mathrm{C}[\mathtt{f}(p_1, \cdots, p_n)\sigma]$ and $v = \mathrm{C}[t\sigma]$.
- $\xrightarrow{*}$ is the reflexive and transitive closure of $\rightarrow$.

Program semantics

Hypothesis

Programs are confluent
A sufficient condition (Huet) is that

1. *A variable appears only one in $p_1, \cdots, p_n$ in any rule*
2. *There are no two left-hand side rule wich are overlapping*

Definition

A function ϕ over constructor domains is computed by a program `f` iff

$$\mathtt{f}(w) \xrightarrow{*} \phi(w)$$

Termination methods

Based on termination orderings

- Polynomial interpretation (*Lankford*)
- Recursive path ordering with status (*Plaisted, Dershowitz, Kamin and Lévy . . .*)

PPO	Product path ordering
LPO	Lexicographic path ordering

- Dependency pairs (*Arts and Giesl*)

Other methods

- Size change principle (*Lee,Jones and ben-Amram*)
- Type systems

Reduction ordering

A reduction ordering $\prec$ is a well-founded term ordering

Closed under context If $t \prec s$ then $C[t] \prec C[s]$

Closed under substitution If $t \prec s$ then $\sigma(t) \prec \sigma(s)$

A reduction ordering is compatible with a program `f` if for each rule $\mathtt{f}(p_1, \cdots, p_n) \to t$

$$t \prec \mathtt{f}(p_1, \cdots, p_n)$$

Theorem

A program is terminating iff it admits a compatible reduction ordering.

Reduction ordering

Theorem

A program is terminating iff it admits a compatible reduction ordering.

Proof.

- If $\mathtt{f}$ is terminating, take $\prec$ to be the transitive closure of $\rightarrow$.
- Conversely, $u \rightarrow v$ such that $u = \mathrm{C}[\mathtt{f}(p_1, \cdots, p_n)\sigma]$ and $v = \mathrm{C}[t\sigma]$. where $\mathtt{f}(p_1, \cdots, p_n) \rightarrow t$.
  We have $v \prec u$ because $\prec$ is closed under context and substitution.
  Termination follows by well-foundedness of $\prec$.

$\square$

Recursive path ordering (RPO)

$$t = f(t_1, \cdots, t_n) \prec_{rpo} g(u_1, \cdots, u_m) = u$$

where $\prec_{\mathcal{F}}$ is a precedence on symbols.

$$\frac{\exists i, t \preceq_{rpo} u_i}{t \prec_{rpo} u}$$

$$\frac{\forall i, t_i \prec_{rpo} \mathrm{g}(u_1, \cdots, u_m) \quad f \prec_{\mathcal{F}} \mathrm{g}}{t \prec_{rpo} u}$$

$$\frac{\forall i, t_i \prec_{rpo} u \quad \{t_1, \cdots, t_n\} \prec_{rpo}^{st(f)} \{u_1, \cdots, u_n\} \quad \mathtt{f} \approx_{\mathcal{F}} \mathtt{g}}{t \prec_{rpo} u}$$

Ordering and constructors

Constructors

$$\frac{\forall i\ u_i \prec_{rpo} \mathtt{f}(t_1, \cdots, t_n)}{\mathbf{c}(u_1, \cdots, u_m) \prec_{rpo} \mathtt{f}(t_1, \cdots, t_n)}\ \mathtt{f} \in \mathcal{F}, \mathbf{c} \in \mathcal{C}$$

- So constructors are the smallest wrt $\prec_{\mathcal{F}}$
- So if u and v are two constructor terms, $u \prec_{rpo} v$ iff u is a subterm of v.

Lemma

The number of n-uplets $v_1, \cdots, v_n$ such that

$$(v_1, \cdots, v_n) \prec_{rpo}^{st(f)} (t_1, \cdots, t_n)$$

is bounded by $\prod_i |t_i|$

Proof.

A term t has $|t|$ subterms. □

Status

How to compare recursive calls ?

Status given to functions :

1. Product
2. Lexicographic

To capture some algorithmic patterns

The product status

$$\mathtt{f}(m_1, \cdots, m_p) \prec^{p}_{rpo} \mathtt{f}(n_1, \cdots, n_p)$$

iff

- $\forall 1 \leq i \leq p, m_i \preceq_{rpo} n_i$
- $\exists 1 \leq j \leq p$ such that $m_j \prec_{rpo} n_j$.

Example

$$\begin{aligned} \text{shuffle}(\epsilon, y) &\rightarrow y \\ \text{shuffle}(x, \epsilon) &\rightarrow x \\ \text{shuffle}(\mathbf{i}(x), \mathbf{j}(y)) &\rightarrow \mathbf{i}(\mathbf{j}(\text{shuffle}(x, y))) \qquad \mathbf{i}, \mathbf{j} \in \{\mathbf{0}, \mathbf{1}\} \end{aligned}$$

The lexicographic extension

$$\mathtt{f}(m_1, \cdots, m_p) \prec^{l}_{rpo} \mathtt{f}(n_1, \cdots, n_p)$$

iff $\exists j$ s.t.

- $\forall i < j, m_i \preceq_{rpo} n_i$
- $m_j \prec_{rpo} n_j$.

Example

$$\begin{aligned}\mathtt{verif}(\textbf{Exists}(x, \phi), \sigma) \rightarrow \mathtt{or}(&\mathtt{verif}(\phi, \text{update}(\sigma, x, \textbf{true})) \\ &\mathtt{verif}(\phi, \text{update}(\sigma, x, \textbf{false})))\end{aligned}$$

Program termination

Theorem
A rpo-ordering on terms is a reduction ordering

Proof.
A consequence of Higmann and Kruskal Theorem. □

Theorem
A program is terminating iff it admits a compatible ordering $\prec_{rpo}$ for some precedence $\prec_{\mathcal{F}}$ and status st. That is for each rule

$$t \prec_{rpo} \mathtt{f}(p_1, \cdots, p_n)$$

Primitive recursion

A $\mathrm{RPO_{Pro}}$-program is a program which terminates by $\prec_{rpo}$ with product comparison status.

Theorem (Hofbauer (92))

The set of RPO_{Pro}-functions,computed by RPO_{Pro}-programs, is exactly the set of primitive recursive functions.

Example

$$
\begin{aligned}
f(\epsilon, \overline{x}) &\rightarrow g(\overline{x}) \\
f(\mathbf{0}(w), \overline{x}) &\rightarrow h_0(w, \overline{x}, f(w, \overline{x})) \\
f(\mathbf{1}(w), \overline{x}) &\rightarrow h_1(w, \overline{x}, f(w, \overline{x})) \qquad \overline{x} = x_1, \ldots, x_n
\end{aligned}
$$

is a $\mathrm{RPO_{Pro}}$-program.

Multiple recursive

A RPO-program which terminates by $\prec_{rpo}$ with any kind of status .

Theorem (Weiermann (95))

The set of functions computed by RPO-programs, is exactly the set of multiple recursive functions.

Example (Ackermann)

$$\begin{aligned} \mathtt{ack}(0, n) &= \mathbf{suc}(n) \\ \mathtt{ack}(m+1, 0) &= \mathtt{ack}(m, 1) \\ \mathtt{ack}(m+1, n+1) &= \mathtt{ack}(m, \mathtt{ack}(m+1, n)) \end{aligned}$$

is a RPO-program because it terminates using a lexicographic status.

Intentionality

Both examples are not *primitive recursive* Tail recursion terminates by lexicographic orders

$$\begin{aligned} \mathrm{reverse}(\epsilon, y) &\to y \\ \mathrm{reverse}(\mathbf{i}(x), y) &\to \mathrm{reverse}(x, \mathbf{i}(y)) \qquad \mathbf{i} \in \{\mathbf{0}, \mathbf{1}\} \end{aligned}$$

Colson's inf terminatesz by product orders

$$\begin{aligned} \mathrm{inf}(0, y) &\to 0 \\ \mathrm{inf}(x, 0) &\to 0 \\ \mathrm{inf}(\mathbf{suc}(x), \mathbf{suc}(y)) &\to \mathbf{suc}(\mathrm{inf}((x, y))) \end{aligned}$$

Term ordering capture a large class of algorithms Polynomial time ordering, see Light MPO (M03).

Assignments

An assignment of f is $(\!| f |\!) : (\mathbb{R}^+)^n \rightarrow \mathbb{R}^+$ satisfying:

- $(\!| f |\!)(X_1, \cdots, X_n) \geq X_i$
- $(\!| f |\!)$ is increasing (not-strictly).
  If $X \leq Y$ then

$$(\!| f |\!)(X_1, \ldots, X, \ldots, X_n) \leq (\!| f |\!)(X_1, \ldots, Y, \ldots, X_n)$$

Term assignment

$$(\!| f(t_1, \cdots, t_n) |\!) = (\!| f |\!)((\!| t_1 |\!), \ldots, (\!| t_n |\!))$$

Quasi interpretations

Definition

An assignment is a *quasi-interpretation* if for any rule $l \to r$,

$$(\!| l\sigma |\!) \geq (\!| r\sigma |\!)$$

(Marion, Moyen, Bonfante)

where $\sigma : \mathit{Variables} \mapsto \mathit{Values}$ is a constructor substitution.

Shuffle

$$\begin{aligned} \text{shuffle}(\epsilon, y) &\to y \\ \text{shuffle}(x, \epsilon) &\to x \\ \text{shuffle}(\mathbf{i}(x), \mathbf{j}(y)) &\to \mathbf{i}(\mathbf{j}(\text{shuffle}(x, y))) \qquad \mathbf{i}, \mathbf{j} \in \{\mathbf{0}, \mathbf{1}\} \end{aligned}$$

- $(\!|\epsilon|\!) = 0$
- $(\!|\mathbf{0}|\!)(X) = (\!|\mathbf{1}|\!)(X) = X + 1$
- $(\!|\text{shuffle}|\!)(X, Y) = X + Y$

$$\begin{aligned} (\!|\text{shuffle}(\mathbf{i}(x), \mathbf{j}(y))|\!) = X + 1 + Y + 1 \geq (\!|\mathbf{i}(\mathbf{j}(\text{shuffle}(x, y)))|\!) \\ = 1 + 1 + X + Y \end{aligned}$$

Max functions

$$\mathtt{max}(\mathbf{0}, y) \to y$$
$$\mathtt{max}(x, \mathbf{0}) \to x$$
$$\mathtt{max}(\mathbf{suc}(x), \mathbf{suc}(y)) \to \mathbf{suc}(\mathtt{max}(x, y))$$

- $(\!|\mathbf{0}|\!) = 0$
- $(\!|\mathbf{suc}|\!)(X) = X + 1$
- $(\!|\mathtt{max}|\!)(X, Y) = \max(X, Y)$

$$\begin{aligned}(\!|\mathtt{max}(\mathbf{suc}(x), \mathbf{suc}(y))|\!) &= \max(X+1, Y+1)\\ &= (\!|\mathbf{suc}(\mathtt{max}(x, y))|\!)\\ &= 1 + \max(X, Y)\end{aligned}$$

Insertion sort

$$
\begin{aligned}
\mathtt{if}\ \mathbf{tt}\ \mathtt{then}\ x\ \mathtt{else}\ y &\to x \\
\mathtt{if}\ \mathbf{ff}\ \mathtt{then}\ x\ \mathtt{else}\ y &\to y \\
\mathbf{0} < \mathbf{suc}(y) &\to \mathbf{tt} \\
x < \mathbf{0} &\to \mathbf{ff} \\
\mathbf{suc}(x) < \mathbf{suc}(y) &\to x < y \\
\mathtt{insert}(a, \epsilon) &\to \mathbf{cons}(a, \epsilon) \\
\mathtt{insert}(a, \mathbf{cons}(b, l)) &\to \mathtt{if}\ a < b \\
&\quad\ \mathtt{then}\ \mathbf{cons}(a, \mathbf{cons}(b, l)) \\
&\quad\ \mathtt{else}\ \mathbf{cons}(b, \mathtt{insert}(a, l)) \\
\mathtt{sort}(\epsilon) &\to \epsilon \\
\mathtt{sort}(\mathbf{cons}(a, l)) &\to \mathtt{insert}(a, \mathtt{sort}(l))
\end{aligned}
$$

QI of Insertion sort

$$(\!|\mathbf{tt}|\!) = (\!|\mathbf{ff}|\!) = (\!|\mathbf{0}|\!) = (\!|\epsilon|\!) = 0$$
$$(\!|\mathbf{suc}|\!)(X) = X + 1$$
$$(\!|\mathbf{cons}|\!)(X, Y) = X + Y + 1$$

And function symbols

$$(\!|\mathtt{if\ \ then\ \ else\ \ }|\!)(X, Y, Z) = \max(X, Y, Z)$$
$$(\!|<|\!)(X, Y) = \max(X, Y)$$
$$(\!|\mathtt{insert}|\!)(X, Y) = X + Y + 1$$
$$(\!|\mathtt{sort}|\!)(X) = X$$

Properties

Proposition

$(\!|-|\!)$ is a quasi-interpretation.
Take two terms u and v such that $u \xrightarrow{*} v$, we have

$$(\!|v|\!) \leq (\!|u|\!)$$

Proof.

$\exists$ a subst. σ and $\mathtt{f}(p_1, \cdots, p_n) \to t$ such that
$u = \mathrm{C}[\mathtt{f}(p_1, \cdots, p_n)\sigma]$ and $v = \mathrm{C}[t\sigma]$.
Since QI are compatible with program rules

$$(\!|t\sigma|\!) \leq (\!|\mathtt{f}(p_1, \cdots, p_n)\sigma|\!)$$

Monotonicity implies

$$(\!|\mathrm{C}[t\sigma]|\!) \leq (\!|\mathrm{C}[\mathtt{f}(p_1, \cdots, p_n)\sigma]|\!)$$

$\square$

Additive QI

- For each symbol f, $(\!|f|\!)$ is bounded by a polynomial
- For each constructor $\mathbf{c}$,

$$(\!|\mathbf{c}|\!)(X_1, \ldots, X_n) = \sum_i X_i + \alpha_{\mathbf{c}} \qquad \text{where } \alpha_{\mathbf{c}} \geq 1$$

Definition (Size)

$$|\mathbf{c}| = 0 \qquad\qquad |\mathbf{c}(t_1, \cdots, t_n)| = 1 + \sum_i |t_i|$$

Proposition

For any constructor term t,

$$|t| \leq (\!|t|\!) \tag{1}$$

$$(\!|t|\!) \leq k \times |t| \tag{2}$$

Addition and multiplication

Domain of unary integers $\{\mathbf{0}, \mathbf{suc}\}$

$$\begin{aligned}
\mathtt{add}(\mathbf{0}, y) &\to y \\
\mathtt{add}(\mathbf{suc}(x), y) &\to \mathbf{suc}(\mathtt{add}(x, y)) \\
\mathtt{mult}(\mathbf{0}, y) &\to \mathbf{0} \\
\mathtt{mult}(\mathbf{suc}(x), y) &\to \mathtt{add}(y, \mathtt{mult}(x, y))
\end{aligned}$$

- $(\!|\mathbf{0}|\!) = 1$
- $(\!|\mathbf{suc}|\!)(X) = X + 1$
- $(\!|\mathtt{add}|\!)(X, Y) = X + Y$
- $(\!|\mathtt{mult}|\!)(X, Y) = X \times Y$

Any polynomial has an additive QI.

Exponential has no additive QI

$$\exp(\mathbf{0}) \to \mathbf{suc}(\diamond)$$
$$\exp(\mathbf{suc}'(x)) \to \mathrm{add}(\exp(x), \exp(x))$$

$$(\!|\mathbf{suc}'|\!)(X) = 2X + 1$$
$$(\!|\exp|\!)(X) = X + 1$$

Fact

There is no additive QI for exp !

Proof.

No polynomial solution.

$$(\!|\exp|\!)(X + \alpha) \geq (\!|\exp|\!)(X) + (\!|\exp|\!)(X) = 2 \times (\!|\exp|\!)(X)$$

$\square$

Exponential time evaluation

Theorem

Assume that f admits an additive QI.
There is an evaluation procedure `eval` *such that*

$$\texttt{eval}(f, t_1, \cdots, t_n) = \begin{cases} w & \texttt{f}(t_1, \cdots, t_n) \xrightarrow{*} w \\ \perp & \textit{otherwise} \end{cases}$$

which runs in $O(2^{\sum_{i=1}^{n} |t_i|^k})$.

Proof.

- The size of each intermediate value is bounded by $(\!| \texttt{f}(t_1, \cdots, t_n) |\!)$
- $\texttt{f}(t_1, \cdots, t_n)$ is computed in space $O((\!| \texttt{f}(t_1, \cdots, t_n) |\!))$ on a TM with an unbounded stack.
- Cook's simulation implies that `eval` runs in $2^{c \times (\!| \texttt{f}(t_1, \cdots, t_n) |\!)}$
- that is runs in $O(2^{\sum_{i=1}^{n} |t_i|^k})$

Characterization of PTIME

A $\mathrm{RPO}^{\mathrm{QI}}_{\mathrm{Pro}}$-program is a $\mathrm{RPO}_{\mathrm{Pro}}$-program, which

1. terminates by $\prec_{rpo}$ with product comparison status,
2. admits an additive quasi-interpretation $(\!|\_|\!)$

Theorem (Marion-Moyen)

The set of functions which are computed by a RPO^{QI}_{Pro}-program is exactly the set PTIME *of functions computable in polynomial time.*

ICAR system implements this resource analysis method. (Moyen)

Additive QI captured PTIME

Lemma

Assume that ϕ is computable in polynomial time.
Then ϕ is computable by a RPO^{QI}_{Pro}-program.

Proof.

A configuration is $\langle q, u, v \rangle$ where

- q is a state,
- u is the left tape
- v is the right tape
- the head is scanning the first letter of u.

$\square$

Additive QI captured PTIME

`state` gives the next configuration

$$\texttt{state}(\langle q, u, v\rangle) = \langle q', u', v'\rangle$$

$$\begin{aligned}\texttt{eval}(0, \langle q, u, v\rangle) &= \langle q, u, v\rangle \\ \texttt{eval}(\mathbf{suc}(t), c) &= \texttt{state}(\texttt{eval}(t, c))\end{aligned}$$

$$\begin{aligned}(\!|\texttt{state}|\!)(X) &= X + 1 \quad (\!|\langle X, Y, Z\rangle|\!) = X + Y + Z + 1 \\ (\!|\texttt{eval}|\!)(T, X) &= T + X \\ (\!|0|\!) &= 0 \qquad (\!|\mathbf{suc}|\!)(X) = X + 1\end{aligned}$$

A polynomial P is computed by a $\mathrm{RPO}^{\mathrm{QI}}_{\mathrm{Pro}}$-program

$$\phi(w) = \texttt{eval}(P(w), \langle q_0, w, \epsilon\rangle)$$

Computation of Additive QI

Lemma

Let $\mathtt{f}$ *be an additive* RPO^{QI}_{Pro}*-program.*
For each constructor term $t_1, \cdots, t_n$*, the runtime to compute* $\mathtt{f}(t_1, \cdots, t_n)$ *is bounded by a polynomial in* $\max_{i=1}^{n} |t_i|$.

Proof.

- We construct a call-by-value interpreter with cache
- We show that it runs within $P(\llparenthesis \mathtt{f}(t_1, \cdots, t_n) \rrparenthesis)$ where P is a polynomial.
- Since the QI is additive $\llparenthesis t_i \rrparenthesis \leq O(|t_i|)$ and $\llparenthesis \mathtt{f}(t_1, \cdots, t_n) \rrparenthesis \leq P(\max_{i=1}^{n} |t_i|)$
- So, runtime evaluation is bounded by a polynomial

$\square$

Memoisation

Example

Computing the length of the longest common subsequence

$$
\begin{aligned}
\texttt{lcs}(x,\epsilon) &\to 0\\
\texttt{lcs}(\epsilon,y) &\to 0\\
\texttt{lcs}(\mathbf{i}(x),\mathbf{i}(y)) &\to \texttt{lcs}(x,y)+1\\
\texttt{lcs}(\mathbf{i}(x),\mathbf{j}(y)) &\to \max(\texttt{lcs}(x,\mathbf{j}(y)),\texttt{lcs}(\mathbf{i}(x),y))
\end{aligned}
$$

- The rewriting calculation required $O(2^n)$ steps
- But, `lcs` terminates by RPO and admits an additive QI : $(\!|\texttt{lcs}|\!) = \max$.
- So, the function computed by `lcs` is polynomial time

CBV with cache

$$\frac{\sigma(x) = w}{\mathcal{E}, \sigma \vdash \langle C, x \rangle \to \langle C, w \rangle} \qquad \frac{\mathbf{c} \in \mathcal{C} \quad \mathcal{E}, \sigma \vdash \langle C_{i-1}, t_i \rangle \to \langle C_i, w_i \rangle}{\mathcal{E}, \sigma \vdash \langle C_0, \mathbf{c}(\bar{t}) \rangle \to \langle C_n, \mathbf{c}(\overline{w}) \rangle}$$

$$\frac{\mathtt{f} \in \mathcal{F} \quad \mathcal{E}, \sigma \vdash \langle C_{i-1}, t_i \rangle \to \langle C_i, w_i \rangle \quad (\mathtt{f}(\overline{w}), w) \in C_n}{\mathcal{E}, \sigma \vdash \langle C_0, \mathtt{f}(t_1, \cdots, t_n) \rangle \to \langle C_n, w \rangle}$$

$$\frac{\begin{array}{c} \mathtt{f}(\overline{p}) \to r \in \mathcal{E} \quad p_i \sigma' = w_i \\ \mathcal{E}, \sigma \vdash \langle C_{i-1}, t_i \rangle \to \langle C_i, w_i \rangle \quad \mathcal{E}, \sigma' \vdash \langle C_n, r \rangle \to \langle C, w \rangle \end{array}}{\mathcal{E}, \sigma \vdash \langle C_0, \mathtt{f}(t_1, \cdots, t_n) \rangle \to \langle C \bigcup (\mathtt{f}(\overline{w}), w), w \rangle}$$

$\langle C, t \rangle \Downarrow \langle C', w \rangle$ means the computation of t is w given an initial cache C.

CBV with cache

Lemma

Let $\mathtt{f}$ be a RPO^{QI}_{Pro}-program. For each constructor term $t_1, \cdots, t_n$, the runtime of the call by value interpreter with cache to compute $\mathtt{f}(t_1, \cdots, t_n)$ is bounded by a polynomial in $(\!| \mathtt{f}(t_1, \cdots, t_n) |\!)$.

Proof.

- We memorize all intermediate function values in cache.
- Time is at most quadratic in the size of the cache.
- Show that the cache size is polynomially bounded in $(\!| \mathtt{f}(t_1, \cdots, t_n) |\!)$.
- Conclusion follows because of additive QI.

$\square$

Size of the cache

- Suppose that $f(t_1, \cdots, t_n)$ is the input
- The number of possible recursive calls

$$\#\{(u_1, \cdots, u_n) \mid (u_1, \cdots, u_n) \prec_{rpo}^{prod} (t_1, \cdots, t_n)\} \leq \prod_i |t_i|$$

- The function f calls g only if $g \prec_{\mathcal{F}} f$.
- If $(g, u_1, \cdots, u_n, u_o)$ is in a cache, then

$$|u_i| \leq (\!| f(t_1, \cdots, t_n) |\!)$$

- Conclusion follows …

Optimization

1. The cache may be minimized. Because the result of $f(t_1, \cdots, t_n$ is not necessary if we know the value of $f(t_1, \cdots, t_n)$ and $u_1, \cdots, u_n \prec_{rpo}^{prod} t_1, \cdots, t_n$.
2. If a recursive call is linear, we do not need to put it in cache.

tail recursion

$\mathtt{f}$ terminates by tail recursion if
(i) the comparison status of $\mathtt{f}$ is lexicographic,
(ii) for each rule $\mathtt{f}(p_1, \cdots, p_n) \to r$ then $\mathtt{f}$ has at most one occurrence in r.

$$\begin{aligned} \mathrm{reverse}(\epsilon, y) &\to y \\ \mathrm{reverse}(\mathbf{i}(x), y) &\to \mathrm{reverse}(x, \mathbf{i}(y)) \end{aligned}$$

Theorem

The set of functions computed by tail recursion programs is exactly PTIME*.*

Characterization of PSPACE

A RPO^{QI}-program is a RPO-program, which

1. terminates by $\prec_{rpo}$ with lexicographic status,
2. admits an additive QI $(\!|\_|\!)$

Theorem (Bonfante, Marion et Moyen)

The set of functions computed by a RPO^{QI}-programs is exactly the set PSPACE *of functions computable in polynomial space.*

Simulate PSPACE computation

- Recurrence with parameter substitution is terminating with a lexicographic status.

$$
\begin{aligned}
f(\epsilon, \overline{y}) &\rightarrow g(\overline{y}) \\
f(\mathbf{0}(x), \overline{y}) &\rightarrow h_0(x, \overline{y}, f(x, \sigma_1(\overline{y})), \ldots, f(x, \sigma_k(\overline{y}))) \\
f(\mathbf{1}(x), \overline{y}) &\rightarrow h_1(x, \overline{y}, f(x, \sigma'_1(\overline{y})), \ldots, f(x, \sigma'_k(\overline{y})))
\end{aligned}
$$

- Polynomials admit a $\mathrm{RPO}^{\mathrm{QI}}_{\mathrm{Lin}}$-program
- We simulate a PRM with forks

RPO^{QI}-programs are PSPACE

Lemma

The space used by a call by value interpreter to compute $f(t_1, \cdots, t_n)$ is bounded by a polynomial in $(\!| f(t_1, \cdots, t_n) |\!)$.

Proof.

Set $A = (\!| \mathtt{f}(t_1, \cdots, t_n) |\!)$

- Each intermediate results are bounded by $O(A)$.
- The maximal length of a branch of a cbv computation is bounded by $\alpha \times A^d$
- Conclusion : the space is at most $O(A^{d+1})$

$\square$

Observations

- Similar result with polynomial interpretation, but less algorithms
- Use of dynamic programming methods (cache) and NLOGSPACE
- Capturing other complexity classes LOGSPACE, ...
- Sup-interpretation (with Péchoux)

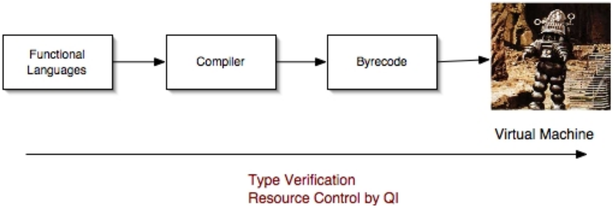
$$
\begin{aligned}
\mathtt{log}(0) &\rightarrow 0 \\
\mathtt{log}(\mathbf{suc}(y)) &\rightarrow \mathbf{suc}(\mathtt{log}(\mathtt{half}(\mathbf{suc}(y)))) \\
\mathtt{half}(0) &\rightarrow 0 \\
\mathtt{half}(\mathbf{suc}(0)) &\rightarrow 0 \\
\mathtt{half}(\mathbf{suc}(y)) &\rightarrow \mathbf{suc}(\mathtt{half}(y))
\end{aligned}
$$

Finding QI

Does a program admits an additive quasi-interpretation ?

- Restrict to $(\max, +, *)$ for candidates to be QI
- Fixed a max-degree
- There is a decision procedure in $2^{|\mathtt{f}|}$
- Because of Tarski's decision procedure for first order polynomial over reals
- NP-hard when considering max-plus assignment on $\mathbb{R}^+$ (Amadio 2003)
- Finding heuristics to synthesis QIs ?
- but, usually the degree is low

A functional Scenario



- We try to find QI for a functional program.
- Then, QI are transfered to bytecode as resource annotations
- Virtual Machine check the resource annotations

From Amadio, Coupet-Grimal, Dal Zilio and Jakubiec

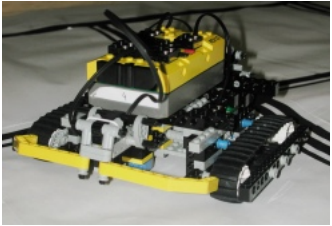
Playing with QI@Nancy

Virtual machine with stack frames

Byte code with 8 instructions

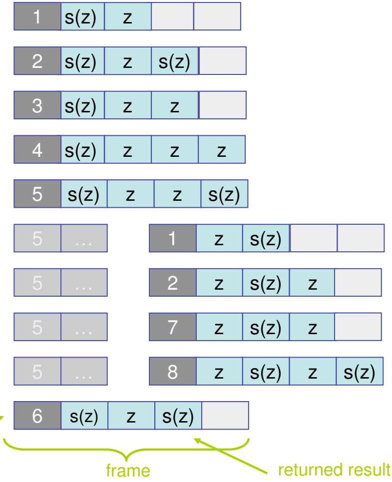
load n	Load the n^{th} argument	call g n	
branch **c**	Conditional & destructor	return	
build **c**	Constructor	stop	
in		out	

Virtual Machine (nearly) runs on Lego Robot Mindstorm



A functional Scenario

1. load 1
2. branch **s** 7
3. load 2
4. build **s** 1
5. call *add* 2
6. return
7. load 2
8. return



Resource bound bytecode certification

At Virtual machine level

- Type verification
- Size control $\rightarrow$ no-malloc at runtime
- Termination

- Application of QI to synchronous cooperative threads (Amadio, Dal Zilio)
- EmBounded and MRG european project (Hofmann)
- Application of QI to control memory allocation of the AVR Butterfly processor (work in progress with Bonfante)

Another applied directions

- Defense against viruses
- and attack by memory-overflow techniques

```
int i;
char buffer[256];

void function(void)
{
for(i=0;i<512;i++)
    buffer[i]='A';
}
```

Fight : Functions vs Algorithms

About functions

- Calculability
- Logical characterization of complexity classes
  - Extensional characterization of complexity classes

About Algorithms

- Studying algorithms : Colson-David, Gurevich, Moschovakis
- Intentional completeness of characterization of complexity classes
  - Intentional characterization of complexity classes
  - including “good” algorithms (Amadio & al, Jones, Hofmann, Bonfante, Marion Moyen, Péchoux)

Some references

R. Amadio.
Max-plus quasi-interpretations.
TLCA 2003, vol 2701 pp 31–45.

R. Amadio, S. Coupet-Grimal, S. Dal-Zilio, and L. Jakubiec.
A functional scenario for bytecode verification of resource bounds.
CSL, vol 3210 LNCS, pp 265–279. 2004

G. Bonfante, A. Cichon, J.-Y. Marion, and H. Touzet.
Algorithms with polynomial interpretation termination proof.
Journal of Functional Programming, 11(1):33–53, 2001.

Guillaume Bonfante, Jean-Yves Marion, and Jean-Yves Moyen.
Quasi-interpretations, a way to control resources.
Theoretical Computer Science, (revision).

J.-Y. Marion.
Complexité implicite des calculs, de la théorie à la pratique.
Habilitation à diriger les recherches, Université Nancy 2, 2000.

J.-Y. Marion and J.-Y. Moyen.
Efficient first order functional program interpreter with time bound certifications.
LPAR 2000, vol 1955 LNCS, pp 25–42