On Sharing, Memoization, and Polynomial Time

Martin Avanzini¹
(Joint work with Ugo Dal Lago¹)

¹Università di Bologna & INRIA, Sophia Antipolis

STACS, March 5, 2015
Implicit Computational Complexity

Characterizing Complexity Classes

L

C

P
Implicit Computational Complexity

Characterizing Complexity Classes
General Simultaneous Recursion (GSR)

• let $A$ be a term algebra formed from constructors $\{c_1, \ldots, c_k\}$

• class of functions definable by general simultaneous recursion (GSR) is least class of function $f : A \times \cdots \times A \to A$:
  1. contains projection and constructor functions
  2. closed under function composition
  3. closed under general simultaneous recursion (GSR)

• functions $f_1, \ldots, f_n$ are defined by GSR with equations

$$f_j(c_i(x_1, \ldots, x_l), \vec{y}) = g_{i,j}(x_1, \ldots, x_l, \underbrace{\vec{f}(x_1, \vec{y}), \ldots, \vec{f}(x_l, \vec{y})}_n)$$

where $\vec{f}(x_i, \vec{y}) = f_1(x_i, \vec{y}), \ldots, f_n(x_i, \vec{y})$
General *Ramified* Simultaneous Recursion (GRSR)

ramification

• take copies $A_0, A_1, A_2, \ldots$ of algebra $A$

• *ramification* can then be expressed as a typing system

\[
\frac{g_{i,j} : A_p^l \times A_n^l \rightarrow A_q}{f_j : A_p \times A \rightarrow A_q} \quad (\text{SimRec})
\]

• functions $f_1, \ldots, f_n$ are defined by GSR with equations

\[
f_j(c_i(x_1, \ldots, x_l), \vec{y}) = g_{i,j}(x_1, \ldots, x_l, \vec{f}(x_1, \vec{y}), \ldots, \vec{f}(x_l, \vec{y}), \vec{y})
\]

where $\vec{f}(x_i, \vec{y}) = f_1(x_i, \vec{y}), \ldots, f_n(x_i, \vec{y})$

[Leivant, 93; Bellantoni & Cook, 92]
GRSR on Trees

- we can define functions \( \text{rabbits}_i : \mathbb{N}_{i+1} \rightarrow T_i \) by

\[
\begin{align*}
\text{rabbits}_i(0) &= B_l \quad b_i(0) = B_l \quad m_i(0) = M_l \\
\text{rabbits}_i(S(n)) &= b_i(n) \quad b_i(S(n)) = B(m_i(n)) \quad m_i(S(n)) = M(m_i(n), b_i(n))
\end{align*}
\]
GRSR on Trees

- we can define functions $\text{rabbits}_i : \mathbb{N}_{i+1} \rightarrow \mathbb{T}_i$ by

  \[
  \text{rabbits}_i(0) = B_l, \quad b_i(0) = B_l, \quad m_i(0) = M_l \\
  \text{rabbits}_i(S(n)) = b_i(n), \quad b_i(S(n)) = B(m_i(n)), \quad m_i(S(n)) = M(m_i(n), b_i(n))
  \]

- data tiering prevents us from defining $\#\text{leafs} : \mathbb{T}_j \rightarrow \mathbb{N}_i$

  \[
  \#\text{leafs}(B_l) = \#\text{leafs}(M_l) = S(0) \\
  \#\text{leafs}(B(t)) = \#\text{leafs}(t) \\
  \#\text{leafs}(M(l, r)) = \text{add}_i(\#\text{leafs}(l), \#\text{leafs}(r))
  \]

  and thus from defining the exponential growing function

  \[
  \text{fib}(n) = \#\text{leafs}(\text{rabbits}_i(n))
  \]
Theorem (Leivant, 93)

The following classes of functions coincide:

1. function over \textit{strings} definable by GRSR \quad \text{constructor arity} \leq 1

2. \textit{class} FPTIME of \textit{polytime computable functions}
Theorem (Leivant, 93)

The following classes of functions coincide:

1. function over *strings* definable by GRSR  
   \[
   \text{constructor arity} \leq 1
   \]

2. class FPTIME of polytime computable functions

- expressive power of GRSR on trees unknown since \( \geq 20 \) years
- does GRSR lead outside FPTIME in general?
Feasible Evaluation of GRSR Functions

The Need for *Sharing*

(a) result of rabbits(6).
Feasible Evaluation of GRSR Functions

The Need for *Sharing*

(a) result of \textit{rabbits}(6).
Feasible Evaluation of GRSR Functions

The Need for *Sharing*

(a) result of *rabbits(6)*.

(b) shared.
Feasible Evaluation of GRSR Functions

The Need for Memoisation

(a) call tree of rabbits(6).
Feasible Evaluation of GRSR Functions

The Need for *Memoisation*

(a) call tree of rabbits(6).
Feasible Evaluation of GRSR Functions

The Need for *Memoisation*

(a) call tree of rabbits(6).

(b) memoized.
Key Observations

Let $f$ be defined by GRSR.
Suppose $f(v_1, \ldots, v_k)$ evaluates to $u$.

Then we can bind by a polynomial in the (shared) size of arguments $v_1, \ldots, v_k$:

1. the shared size of result $u$
   values can always be represented as a compact DAG

2. number of distinct function calls in evaluation of $f(v_1, \ldots, v_k)$
   reduction with memoization feasible

Definition

shared size of value $v :=$ number of distinct subterms in value $v$
Memoization & Sharing, Reconciled
• **configuration** is tuple \((e, C)\)
  
  - \(e\) is expression
  
  - \(C\) is cache, mapping calls \(f(\vec{v})\) to results \(u\)

• semantics are given as **statements**

\[
(f(\vec{v}), C) \downarrow_m (u, D)
\]
Call-by-value Memoizing Semantics

- **configuration** is tuple \((e, C)\)
  - \(e\) is expression
  - \(C\) is cache, mapping calls \(f(\vec{v})\) to results \(u\)

- semantics are given as **statements**
  \[
  (f(\vec{v}), C) \downarrow_m (u, D)
  \]

**Example**

\[
\frac{(f(\vec{v}), u) \in C}{(f(\vec{v}), C) \downarrow_0 (u, C)} \quad \text{(Read)}
\]

\[
\frac{(f(\vec{v}), u') \notin C \quad f(\vec{p}) = r \in E \quad f(\vec{p})\sigma = f(\vec{v}) \quad (r\sigma, C) \downarrow_m (u, D)}{(f(\vec{v}), C) \downarrow_{m+1} (u, D \cup \{(f(\vec{v}), u)\})} \quad \text{(Update)}
\]
Call-by-value Memoizing Semantics

- **configuration** is tuple \((e, C)\)
  - \(e\) is expression
  - \(C\) is cache, mapping calls \(f(\vec{v})\) to results \(u\)

- semantics are given as **statements**

\[
(f(\vec{v}), C) \downarrow_m (u, D)
\]

**Example**

\[
\begin{align*}
(f(\vec{v}), u) &\in C \\
(f(\vec{v}), C) &\downarrow_0 (u, C)
\end{align*}
\]  
(Read)

\[
\begin{align*}
(f(\vec{v}), u') &\not\in C \\
f(\vec{p}) &= r \in \mathcal{E} \\
f(\vec{p})\sigma &= f(\vec{v}) \\
r\sigma &\notin C \\
(f(\vec{v}), C) &\downarrow_{m+1} (u, D \cup \{(f(\vec{v}), u')\})
\end{align*}
\]  
(Update)

- gives rist to a **cost model**, where re-occurring calls are free

  *memoized cost*
Integrating Sharing

crucial, one can now define an implementation such that:

1. each reduction step is atomic
   - no copying of arbitrary large data
   - data is stored on a heap $H$

2. overheads are “small”
Integrating Sharing

crucial, one can now define an implementation such that:

1. each reduction step is atomic
   - no copying of arbitrary large data
   - data is stored on a heap $H$

2. overheads are “small”

implementation is given as reduction relation $\rightarrow_{R_{rsm}}$ on configurations

$$(e, H, C)$$

- $H$ is heap
- $e$ is expression
- $C$ is cache

contain references to heap
Theorem

\((f(\vec{v}), \emptyset) \downarrow_m (u, C) \text{ if and only if } (f(\vec{v}), \emptyset, \emptyset) \rightarrow^n_{\text{Rrsm}} (\ell, H, C')\) where

- \text{result } u \text{ is stored in final heap } H \text{ at location } \ell
- \quad n \leq \delta \cdot m + \text{size}(\vec{v}) \text{ for } \delta \in \mathbb{N}
- \quad \text{size}((\ell, H, C)) \leq \Delta \cdot m + \text{size}(\vec{v}) \text{ for } \Delta \in \mathbb{N}
Polynomial Invariance of Memoized Cost Model

Theorem

\((f(\vec{v}), \emptyset) \downarrow_m (u, C) \text{ if and only if } (f(\vec{v}), \emptyset, \emptyset) \xrightarrow{n}_{\text{Rrs}} (\ell, H, C')\)

where

- result \(u\) is stored in final heap \(H\) at location \(\ell\)
- \(n \leq \delta \cdot m + \text{size}(\vec{v})\) for \(\delta \in \mathbb{N}\)
- \(\text{size}((\ell, H, C)) \leq \Delta \cdot m + \text{size}(\vec{v})\) for \(\Delta \in \mathbb{N}\)

Corollary (Polynomial Invariance of Memoized Cost Model)

There exists a polynomial \(p_f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}\) such that for \((f(\vec{v}), \emptyset) \downarrow_m (u, C)\), the value \(u\) represented as DAG is computable from arguments \(\vec{v}\) in time \(p_f(\text{size}(\vec{v}), m)\).
GRSR is Sound for Polynomial Time

Theorem

Let \( f : \mathbb{A} \rightarrow \mathbb{A}_m \) be a function defined by GRSR.

For all inputs \( \vec{v} \), a DAG representation of \( f(\vec{v}) \) is \textit{computable in time polynomial} in the sizes of the inputs.
GRSR is Sound for Polynomial Time

Theorem
Let $f : \mathbb{A} \to \mathbb{A}_m$ be a function defined by GRSR.
For all inputs $\vec{v}$, a DAG representation of $f(\vec{v})$ is computable in time polynomial in the sizes of the inputs.

Outline.

- by observation on number of distinct function calls during evaluation

$$(f(\vec{v}), \emptyset) \Downarrow_m (u, C) \implies m \leq p_f(\text{size}(\vec{v}))$$

for a polynomial $p_f$
Theorem

Let \( f : A \rightarrow A_m \) be a function defined by \textsf{GRSR}.

For all inputs \( \vec{v} \), a DAG representation of \( f(\vec{v}) \) is \textit{computable in time polynomial} in the sizes of the inputs.

Outline.

- by observation on number of distinct function calls during evaluation
  \[
  (f(\vec{v}), \emptyset) \downarrow_m (u, C) \implies m \leq p_f(\text{size}(\vec{v}))
  \]
  for a polynomial \( p_f \)
- the theorem then follows from polynomial invariance of memoized cost model
1. memoized cost gives rise to notion of **memoized runtime complexity**, this cost model is **polynomial invariant** if we allow sharing

2. **general simultaneous ramified recursion** is sound for **polynomial time**
   - extensions, such as parameter substitution, lead immediately outside polynomial time
Thanks!
Cost Annotated Memoizing Semantics

\[
(f(\vec{v}), v) \in C \\
(\overline{f(\vec{v})}, C) \downarrow (v, C)
\]
(Read)

\[
(f(\vec{v}), u') \not\in C \\
f(p) = r \in \mathcal{E} \\
f(p)\sigma = f(\vec{v}) \\
(\overline{f(\vec{v})}, C) \downarrow (u, D \cup \{(f(\vec{v}), u)\})
\]
(Update)

\[
f \in \mathcal{F} \\
(t_i, C_{i-1}) \downarrow (v_i, C_i) \\
(\overline{f(t_1, \ldots, t_k)}, C_0) \downarrow (v, C_{k+1})
\]
(Split)

\[
c \in \mathcal{C} \\
(t_i, C_{i-1}) \downarrow (v_i, C_i) \\
(\overline{c(t_1, \ldots, t_k)}, C_0) \downarrow (c(\vec{v}), C_k)
\]
(Con)
Cost Annotated Memoizing Semantics

\[
\frac{(f(\vec{v}), v) \in C}{(f(\vec{v}), C) \downarrow_0 (v, C)} \quad \text{(Read)}
\]

\[
\frac{(f(\vec{v}), u') \not\in C \quad f(p) = r \in E \quad f(p)\sigma = f(\vec{v}) \quad (r\sigma, C) \downarrow_m (u, D)}{(f(\vec{v}), C) \downarrow_{m+1} (u, D \cup \{(f(\vec{v}), u)\})} \quad \text{(Update)}
\]

\[
\frac{f \in \mathcal{F}}{(t_i, C_{i-1}) \downarrow_{n_i} (v_i, C_i) \quad (f(\vec{v}), C_k) \downarrow_n (v, C_{k+1})}{(f(t_1, \ldots, t_k), C_0) \downarrow_{n+\sum_{i=1}^k n_i} (v, C_{k+1})} \quad \text{(Split)}
\]

\[
\frac{c \in C}{(t_i, C_{i-1}) \downarrow_{n_i} (v_i, C_i)} \quad \frac{(c(t_1, \ldots, t_k), C_0) \downarrow_{\sum_{i=1}^k n_i} (c(\vec{v}), C_k)}{(c(\vec{v}), C_{k+1})} \quad \text{(Con)}
\]
Small Step Semantics

Memoization and Sharing Reconsiled

\[
(f(\ell), \ell) \in C \\
(E[f(\ell)], H, C) \rightarrow_r (E[\ell], H, C)
\]  \hspace{1cm} \text{(read)}

\[
(f(\ell), \ell) \notin C \quad f(\vec{p}) = r \in \mathcal{E}
\]

“\(f(\ell)\) matches \(f(\vec{p})\) with \(\sigma : \mathcal{V} \rightarrow \text{Loc}_H\)”

\[
(E[f(\ell)], H, C) \rightarrow_R (E[\langle f(\ell), r\sigma \rangle], H, C)
\]  \hspace{1cm} \text{(rule)}

\[
(E[\langle f(\ell), \ell \rangle], H, C) \rightarrow_s (E[\ell], H, C \cup \{(f(\ell), \ell)\})
\]

\[
(H', \ell) = \text{merge}(H, c(\ell))
\]

\[
(E[c(\ell)], H, C) \rightarrow_m (E[\ell], H', C)
\]  \hspace{1cm} \text{(merge)}