# Implicit Computational Complexity
# *in the small*

*Simone Martini*

Dipartimento di Scienze dell'Informazione
*Alma mater studiorum* • Università di Bologna

and
Projet Focus – INRIA Mediterranée

DICE 2010 - Paphos, Cyprus. March 28th 2010

# Implicit Computational Complexity *in the small*

*Simone Martini*

Dipartimento di Scienze dell'Informazione
*Alma mater studiorum* • Università di Bologna

and
Projet Focus – INRIA Mediterranée

DICE 2010 - Paphos, Cyprus. March 28th 2010

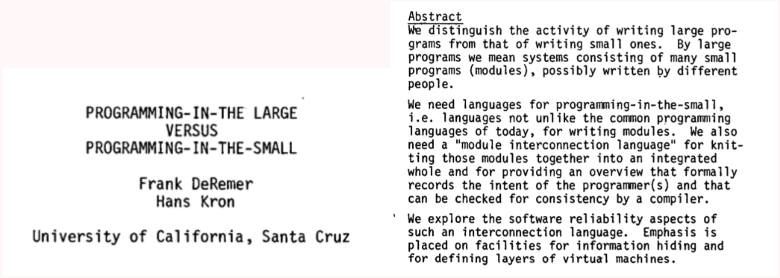
## Outline

## *Implicit* Computational Complexity

- A machine-free, logic-based investigation of the notion of feasible computation
- Feasibility through language restrictions, and not external measure conditions
- Incorporate computational complexity into formal methods in software development and programming language design

## Programming...

International conference on Reliable software, 1975


PROGRAMMING-IN-THE LARGE
VERSUS
PROGRAMMING-IN-THE-SMALL

Frank DeRemer
Hans Kron

University of California, Santa Cruz

Abstract
We distinguish the activity of writing large programs from that of writing small ones. By large programs we mean systems consisting of many small programs (modules), possibly written by different people.

We need languages for programming-in-the-small, i.e. languages not unlike the common programming languages of today, for writing modules. We also need a "module interconnection language" for knitting those modules together into an integrated whole and for providing an overview that formally records the intent of the programmer(s) and that can be checked for consistency by a compiler.

We explore the software reliability aspects of such an interconnection language. Emphasis is placed on facilities for information hiding and for defining layers of virtual machines.

## “We need languages”. . .

Main contribution of a language are

- tools, and
- abstractions

to describe data and control, hiding away unnecessary details.

## Implicit Computational Complexity

- In-the-large: "languages" characterizing complexity classes *e.g.*, Bellantoni-Cook; Leivant; Girard's ligth logics; etc.
- In-the-small: results on specific (machine-free) models of computation, hiding away unnecessary computational details.

## Implicit Computational Complexity

- In-the-large: “languages” characterizing complexity classes *e.g.*, Bellantoni-Cook; Leivant; Girard’s ligth logics; etc.
- In-the-small: results on specific (machine-free) models of computation, hiding away unnecessary computational details.

## ICC “in-the-small”

- Explicitate the complexity content of general results;
- Characterize specific “machine-free” models;
- ...

- To use the result we do not make our hands dirty
- To establish the result we could be very “explicit”.
  I dirty “not my hands only but also my feet and my head!”.

cf. John 13:9

# Part I

## Some results in Recursion Theory

[Asperti, Popl 2008]

## Rice's Theorem

*A set A of indexes of partial recursive functions is extensional iff*

$$i \in A \,\&\, \varphi_i = \varphi_j \Longrightarrow j \in A$$

Theorem (Rice)

*An extensional set $A$ is recursive iff either $A$ is empty, or $A = \mathbb{N}$.*

Useless in complexity, since complexity classes are not extensional.

## Complexity friendly Rice's Theorem

*A set $A$ of indexes of partial recursive functions is* *resource invariant* *iff*

$$i \in A \mathbin{\&} \varphi_i = \varphi_j \mathbin{\&} \textsc{Compl}(i) \in \Theta(\textsc{Compl}(j)) \Longrightarrow j \in A$$

**Theorem (Rice)**

*A resource invariant set $A$ is recursive iff either $A$ is empty, or $A = \mathbb{N}$.*

*Same proof!*

## Rice-Shapiro's Thm

Theorem (Rice-Shapiro)

*If an extensional set $A$ is r.e.,*
*the corresponding set of functions is upward closed and compact.*

Useless in complexity, since complexity classes are not extensional.

## Complexity friendly Rice-Shapiro's Thm

**Theorem (Rice-Shapiro)**

*If a resource invariant set $A$ is r.e.,*
*the corresponding set of functions is upward closed and compact.*

*Same proof, carefully chosen.*

## The ICC in-the-small flavor

- The proof fiddles with the complexity details, but
- The result makes implicit the complexity argument.

*Let $f$ be a specific function.*
*The set of all polynomial programs for $f$, is not recursive, by Rice.*
*I is not r.e., either, because the singleton $\{f\}$ is not compact, or it is not upward closed.*

## The ICC in-the-small flavor

- The proof fiddles with the complexity details, but
- The result makes implicit the complexity argument.

*Let $f$ be a specific function.*
*The set of all polynomial programs for $f$, is not recursive, by Rice.*
*I is not r.e., either, because the singleton $\{f\}$ is not compact, or it is not upward closed.*

## Recursion Thm

**Theorem (II Recursion)**

*Let t be a total recursive function.*
*We can effectively compute an index m such that*

1. $\varphi_m \simeq \varphi_{t(m)}$

## Complexity friendly Recursion Thm

**Theorem (II Recursion)**

*Assume our machine model admits a fair universal machine.*
*Let t be a total recursive function.*
*We can effectively compute an index m such that*

1. $\varphi_m \simeq \varphi_{t(m)}$
2. $\textsc{Compl}(m) \in \Theta(\textsc{Compl}(t(m)))$.

A universal machine $U$ is

1. *fair* if for any $i$ there exists $a$ such that for any $x$, $Time(U(i,x)) \leq a \cdot Time(\varphi_i(x))$.
2. *efficient* if it is fair and the constant $a$ does not depend on the program $i$

[Jones, Popl 1993]

## Intermezzo 1: Fair universal machines?

- Single tape Turing-machines: fair, not efficient (?)
- Multi-tape Turing-machines: not fair (?) folklore
- Jones' fixed alphabet imperative language: efficient [Jones, Popl 1993]
- Lambda-calculus: under which *cost model*?

## Intermezzo 2: How efficient is the fixpoint operator?

- On Turing machines, I do not know...
- On $\lambda$-calculus it takes few $\beta$-reductions:
  1. On call-by-value, weak red:

*For every natural number $n$, there are terms $H_1, \ldots, H_n$ such that for any sequence of values $V_1, \ldots, V_n$ and for any $1 \leq i \leq n$:*

$$H_i V_1 \ldots V_n \rightarrow_v^k V_i(\lambda x.H_1 V_1 \ldots V_n x) \ldots (\lambda x.H_n V_1 \ldots V_n x),$$

*where $k \leq 2n$.*

Part II

## Cost models for rule based languages

## The question

*What is a good cost model for a declarative, rule-based language, taking into account (only) the intrinsic description of that language, and not (also) its implementation on a conventional machine?*

where

*In the intrinsic description of a declarative language, the elementary computation step (e.g., resolution, $\beta$-reduction, firing of a rewrite rule, etc.) is not a constant-time operation.*

## A good cost model. . .

. . . is polynomially related (or *invariant*) to the cost as computed on a Turing machine

*There is a polyomial $p$ such that the cost of computing (the normal form of) $M$ under the cost model $c$ is*

$$\mathrm{Cost}_c(M) \leq p(\mathrm{Cost}_{TM}(M))$$

*For $f$ computed by a Turing machine $\mathcal{M}$ in time $g$, there is a program $N_{\mathcal{M}}$ computing $f$ in $\mathrm{Cost}_c(O(g(n)))$.*

## An answer?

For most such declarative languages, in their generality:

*We do not know.*

because the elementary computation step:

- not only looks non constant time
- but indeed is non constant (or even non poly) time

*Let us look first to $\lambda$-calculus*

## A negative result

- $\lambda$-calculus with full $\beta$-reduction
- implemented as graph reduction (à la Lamping)
  thus realizing Lévy's optimal reduction

- Have a notion of constant-time step

- There are (simply typed) $\lambda$-terms which:
  - normalize in $k$ steps
  - require $\geq O(2^k)$ time on a TM

(On Statman's shoulders:
Asperti and Mairson, POPL 1998; Asperti, Coppola and M., POPL 2000)

## Other cost models?

Are there invariant cost models for full $\beta$-reduction ?

*I do not know of any of them.*

## Restrict the calculus

- Linear $\lambda$-term
  Normalization is PTIME-complete (Mairson, JFP 2004)
  The calculus has little expressivity
- Move to *weak* reductions
  i.e., never reduce *under* a $\lambda$:
  $\lambda x.M$ is always a normal form (in fact, a *value*)

## Weak $\lambda$-calculus

- $\rightarrow_v$ is weak call-by-value
- $\rightarrow_n$ is weak call-by-name

## Weak call-by-value $\lambda$-calculus

- Terms $\quad M ::= x \mid \lambda x.M \mid MM$
- Values $\quad V ::= x \mid \lambda x.M$
- Weak call-by-value reduction

$$\frac{}{(\lambda x.M)V \to_v M\{V/x\}} \qquad \frac{M \to_v N}{ML \to_v NL} \qquad \frac{M \to_v N}{LM \to_v LN}$$

## Explicit representation: the difference cost model

- If terms are represented explicitly as strings
- In particular, we want to print the result as a string

- Difference cost model:
  for each step $M \to_v N$, count $\max\{1, |N| - |M|\}$

*The difference cost model is polynomially invariant.*

(Dal Lago and M., CiE 2006)

## Intermezzo 3

- Under the difference cost model, we have super-efficient interpreters
- Indeed, we have constant overhead interpreters

$$
\begin{aligned}
\{Eval\}(\langle V, U\rangle) &= \{V\}(U) \\
Time(\{Eval\}(\langle V, U\rangle)) &\leq c + Time(\{U\}(V))
\end{aligned}
$$

## Implicit representation: the unitary cost model

- If we allow shared (graph) representation for terms
- In particular, the result could be a shared graph

- Unitary cost model:
  for each step $M \to_v N$, count 1

*The unitary cost model is polynomially invariant.*

(Sands, Gustavsson, and Moran, 2002

Dal Lago and M., ICALP 2009)

## The unitary cost model
## Call-by-name

- Unitary cost model:
  for each step $M \to_n N$, count 1

*The unitary cost model accounts no more than TMs.*

*Most probably, it is also invariant*
*(that is, we can code TMs with a poly overhead).*

(Dal Lago and M., ICALP 2009)

## Why we do not say "It is invariant" ?

In call-by-value:
Given a TM $\mathcal{M}$ in time $g$, code it as $N_{\mathcal{M}}$.
$N_{\mathcal{M}}$ works in time $O(g)$.

Now apply a CPS translation, obtaining $[\![N_{\mathcal{M}}]\!]$
which, in call-by-name, simulates $N_{\mathcal{M}}$,
and work in time. . . ??

## Why we do not say "It is invariant" ?

In call-by-value:
Given a TM $\mathcal{M}$ in time $g$, code it as $N_{\mathcal{M}}$.
$N_{\mathcal{M}}$ works in time $O(g)$.

Now apply a CPS translation, obtaining $[\![N_{\mathcal{M}}]\!]$
which, in call-by-name, simulates $N_{\mathcal{M}}$,
and work in time. . . ??

## A needed ICC in-the-small result

*What is the overhead of a CPS translation ?*

## Intermezzo 4

Let $(\mathcal{A}, \cdot)$ be a combinatory algebra.
For any expression $E$ built on $\mathcal{A}$ and variables, Curry's abstraction algorithm gives an expression $[x]E$ such that

$$[x]E \cdot M \to^* E[M/x]$$

Hence: a translation $(\ )_H$ of $\lambda$-calculus into Combinatory Logic:

$$(\lambda x.M)_H = [x]M_H$$

*What is the overhead of this translation ?*

## Intermezzo 4

Let $(\mathcal{A}, \cdot)$ be a combinatory algebra.
For any expression $E$ built on $\mathcal{A}$ and variables, Curry's abstraction algorithm gives an expression $[x]E$ such that

$$[x]E \cdot M \to^* E[M/x]$$

Hence: a translation $(\ )_H$ of $\lambda$-calculus into Combinatory Logic:

$$(\lambda x.M)_H = [x]M_H$$

*What is the overhead of this translation ?*

## Orthogonal term rewriting

- Signature: for function symbols
- Patterns: terms over functions and variables

- Rules: $f(p_1, \ldots, p_n) \rightarrow_{\equiv} t$
  $f$ is a function symbol; $p_1, \ldots, p_n$ are patterns; $t$ is a (general) term.

- Orthogonal: no rule overlapping; left-linear
- Innermost: the reduced redex does not contain any other redex
- Outermost: the reduced redex is not contained in any other redex

## Orthogonal Term rewriting
## Unitary cost model

- Unitary cost model:
  for each step $t \to s$, count 1

*The unitary cost model is polynomially invariant,*
*both for innermost and outermost reduction.*

(Dal Lago and M., ICALP and FOPARA 2009)

## The proof

From $\lambda$ to constructor rewriting:

$$\begin{tikzcd}
M \arrow[r, "n"', "" , near end] \arrow[d, mapsto, "{[\ ]_\Phi}"'] & \overset{n}{\underset{v}{\phantom{x}}} N \\
{[M]_\Phi} \arrow[r] & {}^{n} t \arrow[u, mapsto, "\langle\ \rangle"']
\end{tikzcd}$$

From constructor rewriting to $\lambda$:

$$\begin{tikzcd}
f(t_1, \ldots, t_h) \arrow[r] \arrow[d, mapsto, "{[\ ]_\Lambda}"', "\langle\!\langle\ \rangle\!\rangle"] & {}^{n}\, u \arrow[d, mapsto] \\
{[f]_\Lambda \langle\!\langle t_1 \rangle\!\rangle \ldots \langle\!\langle t_h \rangle\!\rangle} \arrow[r] & \overset{kn}{\underset{v}{\phantom{x}}} \langle\!\langle u \rangle\!\rangle
\end{tikzcd}$$

## First simulation: From $\lambda$ to constructor rewriting

Idea: full defunctionalization
Any $\lambda$-abstraction becomes a constructor

$$\begin{array}{rcl} [x]_\Phi & = & x; \\ [\lambda x.M]_\Phi & = & \mathbf{c}_{x,M}(x_1,\ldots,x_n), \text{ where } \mathtt{FV}(\lambda x.M) = x_1,\ldots,x_n; \\ [MN]_\Phi & = & \mathbf{app}([M]_\Phi,[N]_\Phi). \end{array}$$

- Constructors: $\mathbf{c}_{x,M}$ for any $M$ and any $x$.
- Functions: **app**.
- Reduction rules:

$$\mathbf{app}(\mathbf{c}_{x,M}(x_1,\ldots,x_n),x) \to [M]_\Phi$$

## Second simulation: From constructor rewriting to $\lambda$

First: Encode *data*, i.e. constructor terms

- Use Scott numerals-like encoding:

$$\begin{array}{rcl} \underline{0} & \equiv & \lambda x_1.\lambda x_2.x_1 \\ \underline{n+1} & \equiv & \lambda x_1.\lambda x_2.\underline{n} \end{array}$$

- Here: $\langle\!\langle\ \rangle\!\rangle_\Lambda$ : constructor terms $\rightarrow$ $\lambda$-terms
  For constructors $\mathbf{c}_1, \ldots, \mathbf{c}_g$:

$$\langle\!\langle \mathbf{c}_i(t_1 \ldots, t_n) \rangle\!\rangle_\Lambda \equiv \lambda x_1. \ldots .\lambda x_g.\lambda y.x_i \langle\!\langle t_1 \rangle\!\rangle_\Lambda \ldots \langle\!\langle t_n \rangle\!\rangle_\Lambda.$$

- $\perp \equiv \lambda x_1. \ldots .\lambda x_g.\lambda y.y$ denotes an error value

## Derivational complexity

*Derivational complexity (= unitary cost model) is an invariant cost model.*

## Main tool
## Term Graph Rewriting

$$a(b(x), y) \rightarrow b(a(y, a(y, x)))$$

$$\begin{tikzcd}
 & \textcircled{a} \arrow[dl] \arrow[dr] & & \boxed{b} \arrow[d] & \\
b \arrow[d] & & \perp & a \arrow[l] \arrow[dr] & \\
\perp & & & & a \arrow[ull, bend left] \arrow[llll, bend left]
\end{tikzcd}$$

## Graph reducibility

For every orthogonal term rewrite system $\mathcal{R}$ over $\Sigma$ and for every term $t$ over $\Sigma$:

Theorem (Outermost Graph-Reducibility)

1. *$t \rightarrow_o^n u$, where $u$ is in normal form; iff*
2. *$[t]_\mathcal{G} \rightarrow_o^m G$, where $G$ is in normal form and $\langle G \rangle_\mathcal{R} = u$.*

*Moreover, $m \leq n$.*

Theorem (Innermost Graph Reducibility)

1. *$t \rightarrow_i^n u$, where $u$ is in normal form; iff*
2. *$[t]_\mathcal{G} \rightarrow_i^n G$, where $G$ is in normal form and $\langle G \rangle_\mathcal{R} = u$.*

## Complexity

**Theorem**

*For every orthogonal, constructor term rewriting system $\mathcal{R}$, there is a polynomial $p : \mathbb{N}^2 \to \mathbb{N}$ such that for every term $t$ the normal form of $[t]_{\mathcal{G}}$ can be computed:*
- *in time at most $p(|t|, \mathrm{Cost}_1(t))$ in outermost reduction;*
- *in time $p(|t|, \mathrm{Cost}_1(t))$ in innermost reduction.*

That is:
derivational complexity is a polynomially invariant cost model for orthogonal constructor term rewriting.

## Part III

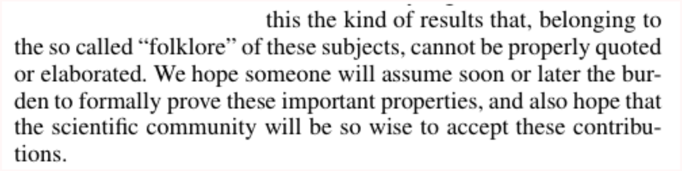
## Conclusions

*General theorems for complexity results, hiding the complexity argument.*

*Logical, abstract notions of computation used as an actual cost measures.*

*Many basic results still missing.*

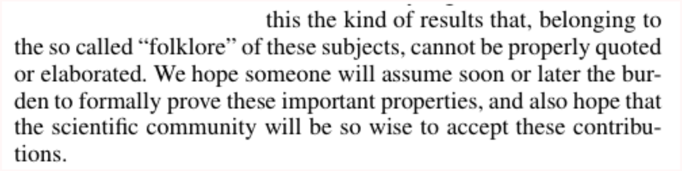
## As a conclusion


this the kind of results that, belonging to the so called "folklore" of these subjects, cannot be properly quoted or elaborated. We hope someone will assume soon or later the burden to formally prove these important properties, and also hope that the scientific community will be so wise to accept these contributions.

[Asperti, Popl 2008]

*Ditto.*

## As a conclusion


this the kind of results that, belonging to the so called "folklore" of these subjects, cannot be properly quoted or elaborated. We hope someone will assume soon or later the burden to formally prove these important properties, and also hope that the scientific community will be so wise to accept these contributions.

[Asperti, Popl 2008]

*Ditto.*