In honor of Cristian Calude's research lead in this area.

A Program-Size Complexity Measure for Mathematical Problems and Conjectures

Michael J. Dinneen

University of Auckland, Auckland, New Zealand

WTCS 2012

Contents

Background

C. S. Calude, E. Calude, M. J. Dinneen. A new measure of the difficulty of problems, *Journal for Multiple-Valued Logic and Soft Computing* 12 (2006), 285–307.

C. S. Calude, E. Calude, M. J. Dinneen. The Complexity of the Riemann Hypothesis, *International Riemann Hypothesis Day*, `http://aimath.org/RH150/rhdayschedule.html`, Auckland, November 18, 2009.

Mathematical problems and conjectures

A problem π of the form $\forall \sigma P(\sigma)$, where P is a computable predicate is called a Π_1–*problem*.

- Any Π_1–problem is finitely refutable.
- For every Π_1–problem $\pi = \forall n P(n)$ we associate the program

$$\sigma_\pi = \inf\{n : P(n) = \text{ false}\}$$

which satisfies:

$$\pi \text{ is true iff } \sigma_\pi \text{ does } \textit{not} \text{ halt.}$$

- Solving the halting problem (for some universal machine) solves all Π_1–problems.

Some known Π_1–problems

Cristian Calude and Elena Calude showed that following problems

- Goldbach's conjecture,
- Fermat's last theorem,
- Riemann hypothesis, and
- the four color theorem

are Π_1–problems.

Of course, not all problems are Π_1–problems.

Is the Collatz' conjecture a Π_1–problem?

Sample Π_1–problem descriptions

- Goldbach's conjecture: every even integer greater than two can be expressed as the sum of two primes.
- Fermat's last theorem: no three positive integers a, b, and c can satisfy the equation $a^n + b^n = c^n$ for any integer value of $n > 2$.
- Collatz' conjecture: for any $n > 0$, the following sequence converges to 1: If n is even, we halve it $(n/2)$, else we do "triple plus one" and get $3n + 1$.
- Riemann hypothesis: all non-trivial zeros of the Riemann zeta function have real part $1/2$.
- The four color theorem: the vertices of any planar graph can be properly colored with 4 colors.

Riemann hypothesis

Theorem

Riemann hypothesis is a Π_1–problem.

The negation of the Riemann hypothesis is equivalent to the existence of positive integers k, l, m, n satisfying the following:

1. $n \geq 600$,
2. $\forall y < n\,[(y+1) \mid m]$,
3. $m > 0 \,\&\, \forall y < m\,[y = 0 \vee \exists x < n\,[\neg\,[(x+1) \mid y]]]$,
4. $\mathit{explog}(m-1, l)$,
5. $\mathit{explog}(n-1, k)$,
6. $(l-n)^2 > 4n^2k^4$,

where $x \mid z$ means "x divides z" and $\mathit{explog}(a, b)$ is the predicate

$$\exists x\,[x > b+1 \,\&\, (1+1/x)^{xb} \leq a+1 < 4(1+1/x)^{xb}].$$

Complexity

Definition

The complexity (with respect to an universal U) of a Π_1–problem π is defined by

$$C_U(\pi) = \inf\{|\sigma_p| \, : \, \pi = \forall n P(n), P \text{ computable}\}.$$

For the universal register machine language U described later we can (somewhat objectively) rank mathematical problems according to their C_U complexities[1]:

- Goldbach's conjecture: 850 $(\mathcal{C}_1)$.
- Fermat's great theorem: 1,738 $(\mathcal{C}_1)$.
- Riemann's hypothesis: 5,170 $(\mathcal{C}_3)$.
- The four color theorem: $\approx$ 7,200 $(\mathcal{C}_4)$.

[1] Here $\mathcal{C}_i$ implies less than i kilobytes needed in a compressed model.

Our (first[2]) universal machine U

The machine U is implemented as a register machine program using five types of instructions that process register data.

By default, all registers, labeled with a string of lower or upper case letters, are initialized to 0. Instructions are labeled by default with 0,1,2,... (in binary).

The register machine instructions will be presented in what follows. Note that in all cases R2 denotes either a label, a register or a binary constant of the form $1(0+1)^*+0$, while R1 and R3 must be a register variable.

The alphabet for programs for U consists of these 14 characters $\{a,\ldots,h,\text{`},',=,\&,+,!,\%\}$, thus each character can be encoded in *four* bits (nibble).

[2]Note newer versions are described in the paper and its references.

=R1,R2,R3 **(EQ R1 R2 R3)**

If the contents of R1 and R2 are equal, then the execution continues at the R3-th instruction, where R3 = 0 denotes the first instruction of the program. If they are not equal, then execution continues with the next instruction in sequence. If the content of R3 is outside the scope of the program, then we have an illegal branch error.

&R1,R2 **(SET R1 R2)**

The contents of register R1 is replaced by the contents of register R2 (or constant R2).

+R1,R2 **(ADD R1 R2)**

The contents of register R1 is replaced by the sum of the contents of registers R1 and R2 (or constant R2).

!R1 **(READ R1)**

One bit is read into the register R1, so the numerical value of R1 becomes either 0 or 1. Any attempt to read past the last data-bit results in a run-time error.

% **(HALT)**

This is the last instruction for each register machine program before the input data. It halts the execution in two possible states: either successfully halts or it halts with an under-read error.

Register machine program

Definition

A *register machine program* consists of a finite list of labeled instructions from the above list, with the restriction that the **HALT** instruction appears only once, as the last instruction of the list.[3]

The input data (a binary string) follows immediately after the **HALT** instruction.

A program not reading the whole data or attempting to read past the last data-bit results in a run-time error.

[3] BTW, this prefix-free property is a required part of a Chaitin Ω_U constant.

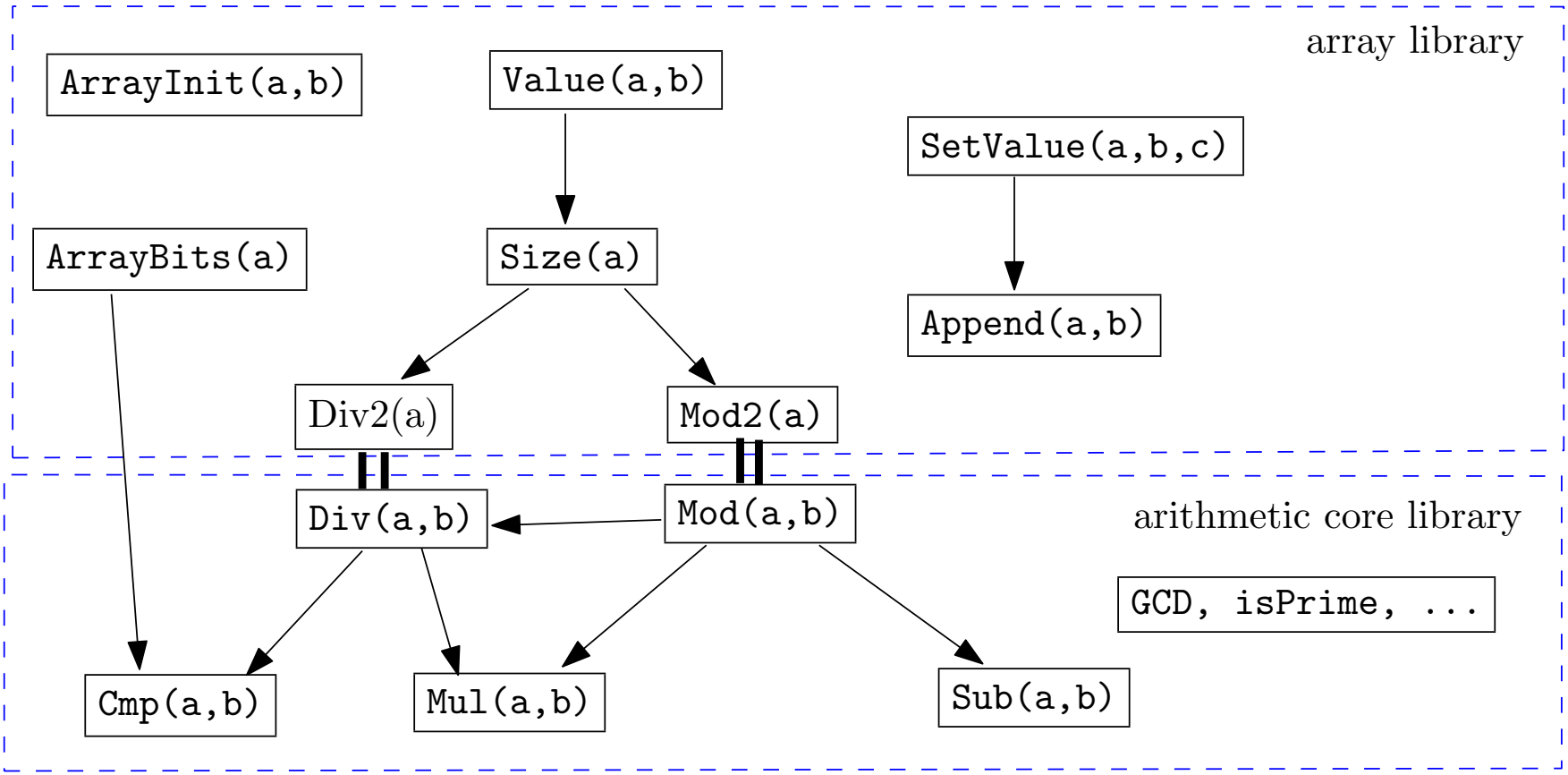
Example: GCD algorithm as register machine

Procedure GCD(int a, int b)
begin
  while $a \neq 0$ **do**
    if $a < b$ **then swap**(a,b)
    $a \leftarrow a - b$
return b
end
GCD(12,8)

```
&a,1100&b,1000&c,1001&d,1100&e,10010&f,10100&g,10110
&h,11010&aa,11100=a,0,aa&ab,a&ac,b=ab,b,f+ab,1+ac,1
=a,ac,f=b,ab,e=d,d,d&b,a&a,ab&ab,b&ac,0=a,ab,h+ab,1
+ac,1=g,g,g&a,ac=c,c,c%
```

Contributions given in my WTCS 2012 paper

Compact subroutines to manipulate arrays with register machines:



Thanks to Y.-B. Kim for assistance in preparing some of these slides.

Definition (Simple P system)

A *simple P system* of order n is a system $\Pi = (O, K, \Delta)$, where:

1. O is a finite non-empty alphabet of *symbols*.
2. $K = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$ is a finite set of *cells*, where each cell $\sigma_i \in K$ is of the form $\sigma_i = (Q_i, s_{i0}, w_{i0}, R_i)$ where
   - Q_i is a finite set of *states*,
   - $s_{i0} \in Q_i$ is *initial state* ($s_i \in Q_i$ is *current state*),
   - $w_{i0} \in O^*$ is *initial content* and ($w_i \in O^*$ is *current content*),
   - R_i is a finite *linearly ordered* set of evolution rules (i.e. transition multiset rewriting rules with *priority*).
     An evolution rule $r \in R_i$ has the form $j\ s\ u \to_\alpha s'\ v$ where:
     - $\alpha \in \{\texttt{min}, \texttt{max}\}$ is a *rewriting* operator of r,
     - $j \in \mathbb{N}$ is the *priority* of r,
     - $s, s' \in Q_i$, where s and s' are *source and target states* of r
     - $u \in O^+$ is the multiset pattern to match,
     - $v \in (O \times \tau)^*$, where target indicator $\tau \in \{\odot, \uparrow, \downarrow, \updownarrow\}$,
3. Δ is an irreflexive and asymmetric relation, representing a set of links between cells with *bidirectional* communication.

Register machines without input data

A subclass $\mathcal{C}'$ of the set $\mathcal{C}$ of register machine programs have no input data (or **READ** instructions), but have only "hard-coded" initial sequence of **SET** instructions as data.

Theorem

The set $\mathcal{C}'$ is Turing complete.

Proof sketch.

We convert any register program $P \in \mathcal{C}$ with data D, viewed as a binary string, to an equivalent program $P' \in \mathcal{C}'$ without data. Let S be a register machine that simulates as its data $P + D$. Encode $P + D$ as two arrays of bits P' and D', as described in the WTCS paper. Make P' from S but instead of reading program and data to index bits in P' and D', which are assign to unused registers. Note I/O errors can be detected by simulator P'.

P systems are Turing complete

For an arbitrary register machine M (without data) with $k \geq 0$ registers and $n \geq 1$ instructions, we show how to build a computationally equivalent simple P system $\Pi = (O, K, \Delta)$:

1. $O = \{r_i \mid 0 \leq i < k\} \cup \{d, t\}$, where symbols $r_0, r_1, \ldots, r_{k-1}$ represent the registers of M and $\{d, t\}$ are distinct symbols.
2. $K = \{\sigma_m\}$. Cell $\sigma_m = (Q, s_{m0}, w_{m0}, R)$, represents register machine M, where:
   - $Q = \{s_i, s'_i \mid 0 \leq i < n\} \cup \{s\}$, where states s_i and s'_i correspond to the i-th instruction of M and state s represent a new "springboard jump" state.
   - $s_{m0} = s_0$, indicates the first instruction, i.e. 0-th instruction.
   - $w_{m0} = \{r_i \mid 0 \leq i < k\}$, indicates the initial content of cell σ_m, where the value of each register r_i corresponds to the multiplicity of r_i minus one (e.g. one symbol denotes zero).
   - R is a set of evolution rules, where states s_i and s'_i correspond to instruction i of M. [Described on next two slides.]
3. $\Delta = \emptyset$.

Instruction: (**SET** r_1 r_2) or (**SET** r_1 const)

- If (**SET** r_1 r_2):
  1. $s_i \; r_{i_1} \rightarrow_{\texttt{max}} s_{i+1}$
  2. $s_i \; r_{i_2} \rightarrow_{\texttt{max}} s_{i+1} \; r_{i_1} \; r_{i_2}$
- If (**SET** r_1 const):
  1. $s_i \; r_{i_1} \rightarrow_{\texttt{min}} s_{i+1} \; r_{i_1}^{\texttt{const}_i+1}$
  2. $s_i \; r_{i_1} \rightarrow_{\texttt{max}} s_{i+1}$

Instruction: (**ADD** r_1 r_2) or (**ADD** r_1 const)

- If (**ADD** r_1 r_2):
  1. $s_i \; r_{i_2} \rightarrow_{\texttt{min}} s_{i+1} \; r_{i_2}$
  2. $s_i \; r_{i_2} \rightarrow_{\texttt{max}} s_{i+1} \; r_{i_1} \; r_{i_2}$
- If (**ADD** r_1 const):
  1. $s_i \; r_{i_1} \rightarrow_{\texttt{min}} s_{i+1} \; r_{i_1}^{\texttt{const}_i+1}$

Instruction: (**HALT**)

- No rules.

Instruction: $(\textbf{EQ}\ r_1\ r_1\ r_3)$, $(\textbf{EQ}\ r_1\ r_2\ r_3)$ or $(\textbf{EQ}\ r_1\ \texttt{const}\ r_3)$

- **If** $(\textbf{EQ}\ r_1\ r_1\ r_3)$**:**
  1. $s_i\ r_{i_3} \rightarrow_{\texttt{max}} s\ r_{i_3}\ t$
- **If** $(\textbf{EQ}\ r_1\ r_2\ r_3)$**:**

  Rules for state s_i*:*
  1. $s_i\ r_{i_1}\ r_{i_2} \rightarrow_{\texttt{max}} s_i'\ r_{i_1}\ r_{i_2}$
  2. $s_i\ r_{i_1} \rightarrow_{\texttt{max}} s_i'\ r_{i_1}\ d$
  3. $s_i\ r_{i_2} \rightarrow_{\texttt{max}} s_i'\ r_{i_2}\ d$

  Rules for state s_i'*:*
  1. $s_i'\ d \rightarrow_{\texttt{max}} s_{i+1}$
  2. $s_i'\ r_{i_3} \rightarrow_{\texttt{max}} s\ r_{i_3}\ t$
- **If** $(\textbf{EQ}\ r_1\ \texttt{const}\ r_3)$**:**
  1. $s_i\ r_{i_1}^{\texttt{const}_i+2} \rightarrow_{\texttt{min}} s_{i+1}\ r_{i_1}^{\texttt{const}_i+2}$
  2. $s_i\ r_{i_1}^{\texttt{const}_i+1} \rightarrow_{\texttt{min}} s\ r_{i_1}^{\texttt{const}_i+1}$
  3. $s_i\ r_{i_1} \rightarrow_{\texttt{min}} s_{i+1}\ r_{i_1}$
  4. $s_i\ r_{i_3} \rightarrow_{\texttt{max}} s\ r_{i_3}\ t$
- **"Springboard" state** s **rules:**
  1. $s\ t^{n+1} \rightarrow_{\texttt{min}} s\ t^{n+1}$
  2. $s\ t^{n} \rightarrow_{\texttt{min}} s_{n-1}$
  3. $s\ t^{n-1} \rightarrow_{\texttt{min}} s_{n-2}$

  $\vdots$

  n+1. $s\ t \rightarrow_{\texttt{min}} s_0$

Open problems

- Is the problem complexity invariant under different register machine models and encodings?
- Can we compute any bits of Omega for a compressed version of register machine language?
- Build a new (and compact) register machine subroutines for other popular simple data structures like dictionaries and graphs.
- Investigate the trade-off between memory efficiently data structures and the sizes of methods to access them.

Conclusions

- Use the size of register machine programs to compare the difficulty of mathematical theorems and statements.
- These "program descriptions" for "problem complexities" are structured to enumerate for a counter-example (finitely refutable property needed).
- Using the known universality of register machines we have a simple proof that P systems are Turing complete.
  (We use only one cell and linear number of states and rules.)
- Lots of research still required!