On the complexity of universal programs

Alain Colmerauer

Laboratoire d'Informatique Fondamentale de Marseille,
CNRS et Universités de Provence et de la Méditerranée

Abstract. This paper provides a framework enabling to define and determine the complexity of various universal programs U for various machines. The approach consists of first defining the complexity as the average number of instructions to be executed by U, when simulating the execution of one instruction of a program P with input x.
To obtain a complexity that does not depend on P or x, we then introduce the concept of an introspection coefficient expressing the average number of instructions executed by U, for simulating the execution of one of its own instructions. We show how to obtain this coefficient by computing a square matrix whose elements are numbers of executed instructions when running selected parts of U on selected data. The coefficient then becomes the greatest eigenvalue of the matrix.
We illustrate the approach using two examples of particularly efficient universal programs: one for a three-symbol Turing Machine (blank symbol not included) with an introspection coefficient of 3 672.98, the other for an indirect addressing arithmetic machine with an introspection coefficient of 26.27.

1 Introduction

For the past several years I have been teaching an introductory course designed to initiate undergraduate students to low level programming. My approach was to start teaching them how to program Turing machines. The main exercise in the course consisted of completing and testing a universal program whose architecture I provided. The results were disappointing, the universal program being too slow for executing sizeable programs. Among others it was impossible to run the machine on its own code, in the sense explained in section 4. In subsequent years, I succeeded in designing considerably more efficient universal programs, even though they became increasingly more complex. These improved programs were capable to execute their own code in reasonable times. For simulating the execution of one of its own instructions, the last program executes an average number of 3 672.98 instructions, or more exactly its introspection coefficient – a key concept introduced in this paper – is equal to 3 672.98.

This paper presents this result in a more general context concerning machines other then Turing machines. It is organized in 5 sections followed by an appendix. The first constitutes this introduction and the last the conclusion. Section 2 introduces the concepts of programmed machine, machine, program,

transition and instruction. Section 3 illustrates those concepts on three examples: Turing machines, Turing machines with internal direction (a useful variant of the previous machines), and an indirect addressing arithmetic machine. In Section 4, the main component of this paper, it is shown how to check the existence of introspection coefficients and how to compute their values. The proof of the theorem presented in that section is fairly lengthy. As most proofs it is omitted by lack of space. A more complete version of the paper, with all proofs, is under preparation. Sections 5 and 6 are devoted to two specially efficient universal programs: the first one, as already mentioned, for a Turing machine, the second one for an indirect addressing arithmetic machine.

We are not aware of other work on the design of efficient universal programs. Let us however mention the well known contributions of M. Minsky [1] and Y. Rogozin [3] in the design of universal programs for Turing machines with very small numbers of states. Surprisingly, they seem particularly inefficient in terms of number of executed instructions.

2 Machines

2.1 Basic definitions

Definition 1 A *programmed machine* is an ordered pair $(\mathcal{M}, P)$, where $\mathcal{M}$ is a *machine* and P a *program* for $\mathcal{M}$.

Definition 2 A *machine* $\mathcal{M}$ is a 5-tuple $(\Sigma, C, \alpha, \omega, \mathcal{I})$, where

- Σ, *the alphabet* of $\mathcal{M}$, is a finite not empty set;
- C, is a set, generally infinite, of *configurations*; the ordered pairs (c, c') of elements of C are called *transitions*;
- α, *the input function*, maps each element x of $\Sigma^\star$ to a configuration $\alpha(x)$;
- ω, *the ouput function*, maps each configuration c to an element $\omega(c)$ of $\Sigma^\star$;
- $\mathcal{I}$, is a countable set of *instructions*, an *intruction* being a set of *compatibles* transitions, i.e., whose first components are all distinct.

Definition 3 A *program* P for a machine $\mathcal{M}$ is a finite subset of the instructions set $\mathcal{I}$ of $\mathcal{M}$, such that the transitions of $\bigcup P$ are compatible.[1] A configuration c of $\mathcal{M}$ is *final for* P, if there exists no transition of $\bigcup P$ starting with c.

2.2 How a machine operates

Let $\mathcal{M} = (\Sigma, C, \alpha, \omega, \mathcal{I})$ be a machine and P a program for $\mathcal{M}$. The operation of the machine $(\mathcal{M}, P)$ is explained by the diagram:

$$\begin{array}{ccccccc} x & & & & & & y \\ \downarrow & & & & & & \uparrow \\ c_0 & \longrightarrow & c_1 & \longrightarrow & c_2 & \cdots & c_{n-1} \longrightarrow c_n \end{array}$$

[1] P being a set of sets $\bigcup P$ denotes the set of elements which are member of at least one element of P and thus the set of transitions involved in program P.

and more precisely by the definition of the following functions[2], where x is a word on Σ and c, c' configurations of C:

$$orbit_{\mathcal{M}}(P,\, x) = \begin{cases} \text{the longest sequence } c_0, c_1, c_2, \ldots \text{ with} \\ c_0 = \alpha(x) \text{ and each } (c_i, c_{i+1}) \text{ an element of } \bigcup P. \end{cases}$$

$$out_{\mathcal{M}}(P,\, x) \;\; = \begin{cases} \nearrow, & \text{if } orbit(P,\, x) \text{ is infinite,} \\ \omega(c_n), & \text{if } orbit(P,\, x) \text{ ends with } c_n \end{cases}$$

and, for dealing with complexity,

$$track_{\mathcal{M}}(P,\, c,\, c') = \begin{cases} \text{the shortest sequence of the form} \\ (c_0, c_1)\,(c_1, c_2)\,(c_2, c_3)\,\ldots\,(c_{n-1}, c_n), \\ \text{with } c = c_0, \text{ each } (c_i, c_{i+1}) \in \bigcup P,\ c_n = c', \text{ if it exists,} \\ \\ \nearrow, \text{ if this shortest sequence does not exist,} \end{cases}$$

$$track_{\mathcal{M}}(P,\, x) \;\; = \begin{cases} \text{the longest sequence of the form} \\ (c_0, c_1)\,(c_1, c_2)\,(c_2, c_3)\,\ldots \\ \text{with } orbit(P,\, x) = c_0, c_1, c_2, \ldots. \end{cases}$$

$$cost_{\mathcal{M}}(P,\, x) \;\; = \begin{cases} \infty, & \text{if } track(P,\, x) \text{ is infinite,} \\ |track(P,\, x)|, & \text{if } track(P,\, x) \text{ is finite,} \end{cases}$$

where $|track(P,\, x)|$ denotes the length of the finite sequence $track(P,\, x)$.

2.3 Simulation of a machine by another

Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be two machines of same alphabet Σ and let $\mathcal{P}_1$ and $\mathcal{P}_2$ be the sets of their programs.

Definition 4 Two programmed machines of the form $(\mathcal{M}_1, P_1)$ and $(\mathcal{M}_2, P_2)$ are *equivalent* if, for all $x \in \Sigma^\star$,

$$\begin{aligned} out_{\mathcal{M}_1}(P_1,\, x) \;\; &= out_{\mathcal{M}_2}(P_2,\, x), \\ cost_{\mathcal{M}_1}(P_1,\, x) &= cost_{\mathcal{M}_2}(P_2,\, x). \end{aligned}$$

Definition 5 The program transformation $f_1 : \mathcal{P}_1 \to \mathcal{P}_2$ *simulates* $\mathcal{M}_1$ on $\mathcal{M}_2$ if, for all $P_1 \in \mathcal{P}_1$, the programmed machines $(\mathcal{M}_1, P_1)$ and $(\mathcal{M}_2, f_1(P_1))$ are equivalent.

3 Examples of machines

3.1 Turing machines

Informally these are classical Turing machines with a bi-infinite tape and instructions written $[q_i, abd, q_j]$, with $d = L$ or $d = R$, meaning : if the machine

[2] Index $\mathcal{M}$ is omitted when there is no ambiguity.

is in state q_i and the symbol read by the read-write head is a, the machine replaces a by b, then moves its head one symbol to the left or the right, depending whether $d = L$ or $d = R$, and change its state to q_j. Initially the entire tape is filled with blanks except for a finite portion which contains the initial input, the read-write head being positioned on the symbol which precedes this input. When there are no more instructions to be executed the machine output the longest word which contains no blank symbols and which starts just after the position of the read-write head.

Formally one first introduces an infinite countable set $\{q_1, q_2, \ldots\}$ of *states* and a special symbol $\mathtt{u}$, *the blank*. For any alphabet word x on an alphabet of the form $\Sigma \cup \{\mathtt{u}\}$, one writes $\cdot x$ for x, with all its beginning blanks erased, and $x\cdot$ for x, with all its ending blanks erased.

Definition 6 A *Turing machine* has a 5-tuple of the form $(\Sigma, C, \alpha, \omega, \mathcal{I})$ where,

- $\Sigma \neq \Sigma_{\mathtt{u}}$, with $\Sigma_{\mathtt{u}} = \Sigma \cup \{\mathtt{u}\}$,
- C is the set of 4-tuples of the form $[q_i, \cdot x, a, y\cdot]$, with q_i being any state, x, y taken from $\Sigma_{\mathtt{u}}^\star$ and a taken from $\Sigma_{\mathtt{u}}$,
- $\alpha(x) = [q_1, \varepsilon, \mathtt{u}, x]$, for all $x \in \Sigma^\star$,
- $\omega([q_i, \cdot x, a, y\cdot])$ is the longest element of $\Sigma^\star$ beginning $y\cdot$,
- $\mathcal{I}$ is the set of instructions denoted and defined, for all sates q_i, q_j and elements a, b of $\Sigma_{\mathtt{u}}$, by
  $[q_i,\, abL,\, q_j] \stackrel{\text{def}}{=} \{([q_i, \cdot xc, a, y\cdot],\, [q_j, \cdot x, c, by\cdot]) \,|\, (x, c, y) \in E\}$,
  $[q_i,\, abR,\, q_j] \stackrel{\text{def}}{=} \{([q_i, \cdot x, a, cy\cdot],\, [q_j, \cdot xb, c, y\cdot]) \,|\, (x, c, y) \in E\}$,
  with $E = \Sigma_{\mathtt{u}}^\star \times \Sigma_{\mathtt{u}} \times \Sigma_{\mathtt{u}}^\star$.

3.2 Turing machines with internal direction

These are a variant of the above described Turing machines with an internal moving head direction whose initial value is equal to left-right. The instructions are written $[q_i,\, abs,\, q_j]$, with $s = +$ or $s = -$, meaning : if the machine is in state q_i and the symbol read by the read-write head is a, the machine replaces a by b, keeps its internal direction or changes it depending whether $s = +$ or $s = -$, moves its read-write head one symbol in the new internal direction, and changes its states to q_j.

Formally we get:

Definition 7 A *Turing machine with internal direction* has a 5-tuple of the form $(\Sigma, C, \alpha, \omega, \mathcal{I})$ where,

- $\Sigma \neq \Sigma_{\mathtt{u}}$, with $\Sigma_{\mathtt{u}} = \Sigma \cup \{\mathtt{u}\}$,
- C is the set of 5-tuples of the form $[d, q_i, \cdot x, a, y\cdot]$, with $d \in \{L, R\}$, q_i being a state, x, y taken from $\Sigma_{\mathtt{u}}^\star$ and a taken from $\Sigma_{\mathtt{u}}$,
- $\alpha(x) = [R, q_1, \varepsilon, \mathtt{u}, x]$, for all $x \in \Sigma^\star$,
- $\omega([d, q_i, \cdot x, a, y\cdot])$ is the longest element of $\Sigma^\star$ beginning $y\cdot$,

– $\mathcal{I}$ is the set of instruction denoted and defined, for all states q_i, q_j, all elements a, b of $\Sigma_{\mathtt{u}}$ and all $s \in \{+,-\}$, by
$[q_i,\, abs,\, q_j] \stackrel{\text{def}}{=}$
$\{([d, q_i, \cdot xc, a, y\cdot],\, [L, q_j, \cdot x, c, by\cdot]) \mid\ (d, s) \in E_1 \text{ and } (x, c, y) \in F\} \cup$
$\{([d, q_i, \cdot x, a, cy\cdot],\, [R, q_j, \cdot xb, c, y\cdot]) \mid\ (d, s) \in E_2 \text{ and } (x, c, y) \in F\}$,
with $E_1 = \{(L, +), (R, -)\}$, $E_2 = \{(R, +), (L, -)\}$ and $F = \Sigma_{\mathtt{u}}^{\star} \times \Sigma_{\mathtt{u}} \times \Sigma_{\mathtt{u}}^{\star}$.

3.3 Relationship between the two types of Turing machines

Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be machines of same alphabet Σ, the first one of type Turing and the second one of type Turing with internal direction, and let $\mathcal{P}_1$ and $\mathcal{P}_2$ denote the sets of their programs. As we will see, there exists a strong relation between these two types of machines. First it can be shown that:

Property 1 *The program transformations $g_1 : \mathcal{P}_1 \to \mathcal{P}_2$ and $g_2 : \mathcal{P}_2 \to \mathcal{P}_1$, defined below, simulate $\mathcal{M}_1$ on $\mathcal{M}_2$ and $\mathcal{M}_2$ on $\mathcal{M}_1$:*

$$g_1(P_1) \stackrel{def}{=} \{[q_{2i-h},\, abd,\, q_{2j-k}] \mid [q_i,\, abs,\, q_j] \in P' \text{ and } (h, k, d, s) \in E\},$$
$$g_2(P_2) \stackrel{def}{=} \{[q_{2i-h},\, abs,\, q_{2j-k}] \mid [q_i,\, abd,\, q_j] \in P \text{ and } (h, k, s, d) \in E\},$$

with

$$E = \left\{ \begin{matrix} (1,\, 1,\, +, R) \\ (1,\, 0,\, -, L) \end{matrix} \right\} \cup \left\{ \begin{matrix} (0,\, 1,\, -, R) \\ (0,\, 0,\, +, L) \end{matrix} \right\}.$$

Let us now introduce the following concepts concerning a program P_ℓ for $\mathcal{M}_\ell$, with $\ell = 1$ or $\ell = 2$.

Definition 8 A state q_i is *reachable in P_ℓ*, if $i = 1$ or if there exists a subset of P_ℓ of the form

$$\{[q_{k_0},\, a_1b_1e_1,\, q_{k_1}], [q_{k_1},\, a_2b_2e_2,\, q_{k_2}], \ldots, [q_{k_{m-1}},\, a_mb_me_m,\, q_{k_m}]\},$$

with $k_0 = 1$ ank $k_m = i$, the a_i's and b_i's being taken from Σ and the e_i's being taken from $\{L, R\}$ or $\{+,-\}$, depending on whether $\ell = 1$ or $\ell = 2$.

Definition 9 For all subsets of P_ℓ of the form

$$\{[q_{k_0},\, a_1b_1e_1,\, q_{k_1}], [q_{k_1},\, a_2b_2e_2,\, q_{k_2}], \ldots, [q_{k_{m-1}},\, a_mb_me_m,\, q_{k_m}]\},$$

with the a_i's and b_i's taken from Σ and the e_i's from $\{L, R\}$ or $\{+,-\}$, depending whether $\ell = 1$ or $\ell = 2$, the sequence of 3-tuples $a_1b_1e_1, a_2b_2e_2, \ldots, a_mb_me_m$ is called *a potential effect* of state q_{k_0}. Two states occurring in P which have the same set of potential effects are said to be *mergeable*.

Definitions 10

– *reachable(P_ℓ) denotes the set of instructions of P_ℓ involving reachable states of P_ℓ,*

- *merged*(P_ℓ) *denotes the program obtained by replacing in* P*, and in parallel, every occurrence of a state* q_i *by the state* q_j *of smallest index such that* q_i *and* q_j *are mergeable,*
- *compact*(P_ℓ) *denotes the program obtained by renaming each state* q_i *of* P_ℓ *by* $q_{i'}$ *in such a way that the integers* i' *are as small as possible and that* $i < j$ *entails* $i' < j'$*,*
- $clean(P_\ell) = compact(reachable(merged(P_\ell)))$.

One proves that for $\ell = 1$ and $\ell = 2$:

Property 2 *The programmed machines* $(\mathcal{M}_\ell, clean(P_\ell))$ *and* $(\mathcal{M}_\ell, P_\ell)$ *are equivalent.*

One also proves that:

Theorem 1 *The program transformations* $f_1 : \mathcal{P}_1 \to \mathcal{P}_2$ *and* $f_2 : \mathcal{P}_2 \to \mathcal{P}_1$*, defined by* $f = clean \circ g_\ell$*, with* g_ℓ *defined in Property 1, simulate* $\mathcal{M}_1$ *on* $\mathcal{M}_2$ *and* $\mathcal{M}_2$ *on* $\mathcal{M}_1$ *and are such that* $f_2 \circ f_1 \circ f_2 \circ f_1 = f_2 \circ f_1$ *and* $f_1 \circ f_2 \circ f_1 \circ f_2 = f_1 \circ f_2$.

3.4 Indirect addressing arithmetic machine

This is a machine with an infinity of registers $r_0, r_1, r_2, \ldots$. Each register contains an unbounded natural integer. Each instruction starts with a number and the machine always executes the instruction whose number is contained in r_0 and, except in one case, increases r_0 by 1. There are five types of instructions: assigning a constant to a register, addition and subtraction of a register to/from another, two types of indirect assignment of a register to another and zero-testing of a register content.

More precisely and in accordance with our definition of a machine:

Definition 11 An indirect addressing arithmetic machine has a 5-tuple of the form $(\Sigma, C, \alpha, \omega, \mathcal{I})$, where,

- $\Sigma = \{c_1, \ldots, c_m\}$,
- C is the set of infinite sequences $r = (r_0, r_1, r_2, \ldots)$ of natural integers,
- $\alpha(a_1 \ldots a_n) = (0, 25, 1, \ldots, 1, r_{24+1}, \ldots, r_{24+n}, 0, 0, \ldots)$, with r_{24+i} equal to $1, \ldots, m$ depending whether a_i equals $c_1, \ldots, c_m$,
- $\omega(r_0, r_1, \ldots) = a_1 \ldots a_n$, with a_i equal to $c_1, \ldots, c_m$ depending whether $\overline{r_{r_1+i}}$ equals $1, \ldots, m$, and n being is the greatest integer such that $r_{r_1}, \ldots, r_{r_1+n}$ are elements of $\{1, \ldots, m\}$,
- $\mathcal{I}$ is the set of instructions denoted and defined, for all natural integers i, j, k, by:

$$[i, cst, j, k] \stackrel{\text{def}}{=} \{(r,\, s') \in C^2 \,|\, r_0 = 1,\, s_j = k \text{ and } s_i = r_i \text{ elsewhere}\},$$
$$[i, plus, j, k] \stackrel{\text{def}}{=} \{(r,\, s') \in C^2 \,|\, r_0 = 1,\, s_j = r_j + r_k \text{ and } s_i = r_i \text{ elsewhere}\},$$
$$[i, sub, j, k] \stackrel{\text{def}}{=} \{(r,\, s') \in C^2 \,|\, r_0 = 1,\, s_j = r_j \dot{-} r_k \text{ and } s_i = r_i \text{ elsewhere}\},$$
$$[i, from, j, k] \stackrel{\text{def}}{=} \{(r,\, s') \in C^2 \,|\, r_0 = 1,\, s_j = r_{r_k} \text{ and } s_i = r_i \text{ elsewhere}\},$$
$$[i, to, j, k] \stackrel{\text{def}}{=} \{(r,\, s') \in C^2 \,|\, r_0 = 1,\, s_{r_j} = r_k \text{ and } s_i = r_i \text{ elsewhere}\},$$
$$[i, ifze, j, k] \stackrel{\text{def}}{=} \{(r,\, s') \in C^2 \,|\, r_0 = 1,\, s_0 = \begin{cases} r_k, \text{if } r_j = 0, \\ r_0, \text{if } r_j \neq 0 \end{cases} \text{ and } s_i = r_i \text{ elw.}\},$$

with s' equal to s except that $s'_0 = s_0 + 1$ and with $r_j \dot{-} r_k = \max\{0, r_j - r_k\}$.

4 Universal programs and codings

4.1 Introduction

Let $\mathcal{M} = (\Sigma, C, \alpha, \omega, \mathcal{I})$ be a machine and let us code each program P for $\mathcal{M}$ by a word $code(P)$ on Σ.

Definition 12 The pair $(U,\, code)$, the program U and the coding function $code$, are said to be *universal* for $\mathcal{M}$, if, for all programs P of $\mathcal{M}$ and for all $x \in \Sigma^\star$,

$$\boxed{out(U,\, code(P) \cdot x) = out(P,\, x)}. \tag{1}$$

If in the above formula we replace P by U, and x by $code(U)^n \cdot x$ we obtain:

$$out(U,\, code(U)^{n+1} \cdot x) = out(U,\, code(U)^n \cdot x)$$

and thus:

Property 3 *If $(U,\, code)$ is a universal pair, then for all $n \geq 0$ and $x \in \Sigma^\star$,*

$$\boxed{out(U,\, code(U)^n \cdot x) = out(U,\, x)}. \tag{2}$$

4.2 Complexity and introspection coefficient

Let $(U, code)$ be a universal pair for the machine $\mathcal{M} = (\Sigma, C, \alpha, \omega, \mathcal{I})$. The *complexity* of this pair is the average number of transitions performed by U for producing the same effect as a transition of the program P occurring in the input of U. More precisely:

Definition 13 Given a program P for $\mathcal{M}$ and a word x on Σ with $cost(P,\, x) \neq \infty$, the *complexity* of $(U,\, code)$ is the real number denoted and defined by

$$\boxed{\lambda(P,\, x) = \frac{cost(U,\, code(P) \cdot x)}{cost(P,\, x)}.}$$

The disavantage of this definition is that the complexity depends on the input of U. For an intrinsic complexity, independ of the input of U, we introduce the *introspection coefficient* of $(U, code)$ whose definition is justified by Property 2:

Definition 14 If for all $x \in \Sigma^\star$, with $cost(U,\, x) \neq \infty$, the real number

$$\boxed{\lim_{n\to\infty} \lambda(U,\, code(U)^n \cdot x) = \lim_{n\to\infty} \frac{cost(U, code(U)^{n+1} \cdot x)}{cost(U, code(U)^n \cdot x)}}$$

exists and does not depend on x, then this real number is the *introspection coefficient* of the universal pair $(U, code)$.

4.3 Keeping the same complexity and introspection coefficient

Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be two machines, with same alphabet Σ, let $\mathcal{P}_1$ and $\mathcal{P}_2$ be the sets of their programms, let $f_1 : \mathcal{P}_1 \to \mathcal{P}_2$ and $f_2 : \mathcal{P}_2 \to \mathcal{P}_1$ be to program transformation and let $(U_2, code_2)$ be a universal pair for $\mathcal{M}_2$. Let us make the following hypothesis:

Hypothesis 1 *The transformation f_1 simulates $\mathcal{M}_1$ on $\mathcal{M}_2$, the transformation f_2 simulates $\mathcal{M}_2$ on $\mathcal{M}_1$ and $code_2 \circ f_1 \circ f_2 = code_2$.*

From Theorem 1 at page 6 it follows:

Property 4 *In the particular case where $\mathcal{M}_1$ is a Turing machine and $\mathcal{M}_2$ a Turing with internal direction, Hypothesis 1 is satisfied by taking the transformations f_1 and f_2 of Theorem 1 and by taking $code_2$ of the form $code'_2 \circ f_1 \circ f_2$, with $code'_2$ a mapping of type $\mathcal{P}_2 \to \Sigma^\star$.*

For the sequel, let x, P_1, P_2 respectivley denote an element of $\Sigma^\star, \mathcal{P}_1, \mathcal{P}_2$. We have the following property:

Property 5 *With Hypothesis 1 and if $P_1 = f_2(P_2)$ or $P_2 = f_1(P_1)$, the pair $(U_1,\, code_1)$, with $U_1 = f_2(U_2)$ and $code_1 = code_2 \circ f_1$,*

1. *is universal for $\mathcal{M}_1$,*
2. *has a complexity $\lambda_1(P_1, x)$, undefined or equal to the complexity $\lambda_2(P_2, x)$ of the pair $(U_2, code_2)$, depending whether the real number $\lambda_2(P_2, x)$ is undefined or defined,*[3]
3. *admits the same introspection coefficient as $(U_2, code_2)$ or does not admit an introspection coefficient, depending whether $(U_2, code_2)$ admits or does not admit one,*
4. *is such that $code_1(P_1) = code_2(P_2)$.*

[3] Of course $\lambda_i(P_i, x) = \frac{cost_{\mathcal{M}_i}(U_i,\, code_i(P_i)\cdot x)}{cost_{\mathcal{M}_i}(P_i,\, x)}$.

Proof Let us first prove claim 4. If $P_2 = f_1(P_1)$, since $code1 = code_2 \circ f_1$, we have

$$code_1(P_1) = code_2(f_1(P_1)) = code(P_2).$$

If $P_1 = f_2(P_2)$, since $code_2 = code_2 \circ f_1 \circ f_2$ and $code_2 \circ f_1 = code_1$, we have $code_2 = code_1 \circ f_2$ and thus

$$code_2(P_2) = code_1(f_2(P_2)) = code_1(P_1).$$

Claim 1 is proven by the equalities below, where Q_1 is any element of $\mathcal{P}_1$:

$$\begin{array}{ll}
out_{\mathcal{M}_1}(Q_1, x) \ = & \text{(since } f_1 \text{ simulates } \mathcal{M}_1 \text{ on } \mathcal{M}_2\text{)} \\
out_{\mathcal{M}_2}(f_1(Q_1), x) \ = & \text{(since } (U_2, code_2) \text{ is universal for } \mathcal{M}_2\text{)} \\
out_{\mathcal{M}_2}(U_2,\ code_2(f_1(Q_1))\cdot x) \ = & \text{(since } f_2 \text{ simulates } \mathcal{M}_2 \text{ on } \mathcal{M}_1\text{)} \\
out_{\mathcal{M}_1}(f_2(U_2), code_2(f_1(Q_1))\cdot x) \ = & \text{(since } f_2(U_2) = U_1 \text{ and } code_2 \circ f_1 = code_1\text{)} \\
out_{\mathcal{M}_1}(U_1,\ code_1(Q_1)\cdot x). &
\end{array}$$

Claim 2 is proven by showing the equality of pairs

$$\begin{bmatrix} cost_{\mathcal{M}_1}(U_1,\ code_1(P_1)\cdot x) \\ cost_{\mathcal{M}_1}(P_1,\ x) \end{bmatrix} = \begin{bmatrix} cost_{\mathcal{M}_2}(U_2,\ code_2(P_2)\cdot x) \\ cost_{\mathcal{M}_2}(P_2,\ x) \end{bmatrix}. \tag{3}$$

First of all,

$$\begin{array}{ll}
cost_{\mathcal{M}_1}(U_1, code_1(P_1)\cdot x) \ = & \text{(since } U_1 = f_2(U_2),\ code_1(P_1) = code_2(P_2)\text{)} \\
cost_{\mathcal{M}_1}(f_2(U_2), code_2(P_2)\cdot x) \ = & \text{(since } f_2 \text{ simulates } \mathcal{M}_2 \text{ on } \mathcal{M}_1\text{)} \\
cost_{\mathcal{M}_2}(U_2, code_2(P_2)\cdot x). &
\end{array}$$

Secondly, since f_2 simulates $\mathcal{M}_2$ on $\mathcal{M}_1$, if $P_1 = f_2(P_2)$, and since f_2 simulates $\mathcal{M}_1$ on $\mathcal{M}_2$, if $P_1 = f_2(P_2)$, we have

$$cost_{\mathcal{M}_1}(P_1, x) = cost_{\mathcal{M}_2}(P_2, x).$$

Finally claim 3 is proven by the sequence of equalities:

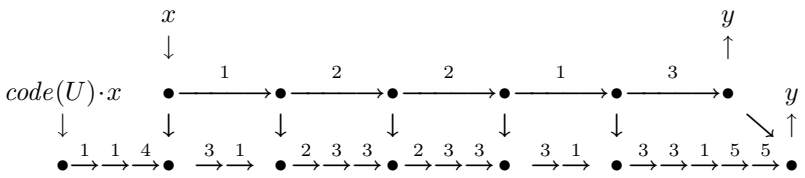
$$\begin{array}{ll}
\begin{bmatrix} cost_{\mathcal{M}_1}(U_1,\ code_1(U_1)\cdot code_1(U_1)^n\cdot x) \\ cost_{\mathcal{M}_1}(U_1,\ code_1(U_1)^n\cdot x) \end{bmatrix} \ = & \text{(by replacing } P_1 \text{ by } U_1 \text{ and} \\
& x \text{ by } code_1(U_1)^n\cdot x \text{ in (3))} \\
\begin{bmatrix} cost_{\mathcal{M}_2}(U_2,\ code_2(U_2)\cdot code_1(U_1)^n\cdot x) \\ cost_{\mathcal{M}_1}(U_2,\ code_1(U_1)^n\cdot x) \end{bmatrix} \ = & \text{(since } code_1(U_1) = code_2(U_2)\text{)} \\
\begin{bmatrix} cost_{\mathcal{M}_2}(U_2,\ code_2(U_2)\cdot code_2(U_2)^n\cdot x) \\ cost_{\mathcal{M}_2}(U_2,\ code_2(U_2)^n\cdot x) \end{bmatrix}. &
\end{array}$$

This completes the proof.

4.4 Existence and value of the introspection coefficient

Let $(U, code)$ be a universal pair for a machine $\mathcal{M} = (\Sigma, C, \alpha, \omega, \mathcal{I})$. Given a word x on Σ, we assume that the computation of the word y by $y = out(U, x)$ can be

synchronized with the computation of the same word y by $y = out(U,\, code(U) \cdot x)$, according to the following diagram:



More precisely we make the hypothesis:

Hypothesis 2 *There exists*

- a synchronization function $\Phi : C_U \to C_U$, *with* $\Phi(c)$ *final for* U *when* c *is final for* U,
- a labelling function $\mu : (\bigcup U)^\star \to (1..n)^\star$ *with* n *a positive integer and* μ *being surjective, such that* $\mu(t_1 \dots t_k) = \mu(t_1) \dots \mu(t_k)$, *for all* $t_1 \dots t_k \in (\bigcup U)^\star$,
- an initial sequence of labels $\delta \in (1..n)^\star$, *independent of* x, *such that*

$$\delta = \mu(track(U,\, \alpha(code(U) \cdot x),\, \Phi(\alpha(x)))),\ \textit{for all } x \in \Sigma^\star,$$

- a label rewriting $\varphi : 1..n \to (1..n)^\star$ *such that, for all* $(c, c') \in \bigcup U$,

$$\varphi(\mu(c, c')) = \mu(track(U,\, \Phi(c),\, \Phi(c'))).$$

We then introduce the column vector B and the square matrix A:

$$B = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}, \qquad b_i = \text{number of occurrences of } i \text{ in } \delta,$$
$$A = \begin{bmatrix} a_{11} \cdots a_{nn} \\ \vdots \qquad \vdots \\ a_{1n} \cdots a_{nn} \end{bmatrix}, \; a_{ij} = \text{number of occurrences of } i \text{ in } \varphi(j). \tag{4}$$

and we conclude by the main theorem of the paper:

Theorem 2 *If the matrix* A *admits a real eigenvalue* λ, *whose multiplicity is equal to* 1 *and whose value is strictly greater than* 1 *and strictly greater then the absolute value* λ' *of the other eigenvalues of* A *then:*

- *the limit matrix* $\bar{A} = \lim_{n\to\infty}(\frac{1}{\lambda}A)^n$ *exists and has at least one non-zero element,*
- *if* $||\bar{A}B|| \neq 0$, *the introspection coefficient of* U *exists and is equal to* λ,
- *if* ℓ *is a real number such that* $\lambda' < \ell < \lambda$ *and the* X_i*'s column vectors defined by* $X_0 = B$ *and* $X_{n+1} = \frac{1}{\ell}AX_n$, *then, when* $n \to \infty$,

$$||X_n|| \to \begin{cases} 0, & \textit{if } ||\bar{A}B|| = 0, \\ \infty, & \textit{if } ||\bar{A}B|| \neq 0. \end{cases}$$

Here $||X||$ denotes the sum of the components of X.

5 Our universals pairs for the two types of Turing machines

5.1 Introduction

We now present two particularly efficient universal pairs, $(U_1, code_1)$, $(U_2, code_2)$, one for the Turing machine $\mathcal{M}_1$ with alphabet $\Sigma = \{\mathtt{o}, \mathtt{i}, \mathtt{z}\}$ and the other for the Turing machine with internal direction $\mathcal{M}_2$ and same alphabet Σ.

The pair $(U_1, code_1)$ is built from the pair $(U_2, code_2)$ by taking $U_1 = f_2(U_2)$ and $code_1 = code_2 \circ f_1$, where f_1 and f_2 are the program transformations introduced in Theorem 1 at page 6. Since $code_2$ will be of the form $code'_2 \circ f_1 \circ f_2$, according to Properties 4 and 5, the pair $(U_1, code_1)$ is indeed universal for $\mathcal{M}_1$ and has the same complexity properties as the pair $(U_2, code_2)$.

Let us mention that U_1 has 361 instructions and 106 states while U_1 has only 184 instructions and 54 states. It remains to present the pair $(U_2, code_2)$.

5.2 Coding function of the universal pair $(U_2, code_2)$

In order to assign a position to each instruction $[q_i, abs, q_j]$ of a program for $\mathcal{M}_2$, we first introduce the numbers:

$$\pi(i,a) = 4(i-1) + \begin{cases} 1, \text{ if } a = \mathtt{u} \\ 2, \text{ if } a = \mathtt{o} \\ 3, \text{ if } a = \mathtt{i} \\ 4, \text{ if } a = \mathtt{z} \end{cases}, \qquad \pi(i) = \tfrac{1}{2}(f(i,\mathtt{o}) + f(i,\mathtt{i})),$$

defined for any positive integer i and any symbol $a \in \{\mathtt{u}, \mathtt{o}, \mathtt{i}, \mathtt{z}\}$. Then let P_2 be a program for $\mathcal{M}_2$ and let $P'_2 = f_1(f_2(P_2))$. We take $code_2(P_2) = code'_2(P'_2)$, with $code'_2(P'_2)$ the word on $\{\mathtt{o}, \mathtt{i}, \mathtt{z}\}$

$$\mathtt{z}I_{4n}\mathtt{z}\ldots\mathtt{z}I_{k+1}\mathtt{z}I_k\mathtt{z}I_{k-1}\mathtt{z}\ldots\mathtt{z}I_1\mathtt{zoi}\ldots\mathtt{izz},$$

where n is the number of states of P'_2, where the size of the *shuttle* $\mathtt{oi}\ldots\mathtt{iz}$ is equal to the longest size of the I_k's minus 5, and where, for all $a \in \Sigma_{\mathtt{u}}$ and $i \in 1..n$,

$$I_{\pi(i,a)} = \begin{cases} \overline{[q_i, abs, q_j]}, \text{ if there exists } b, s, j \text{ with } [q_i, abs, q_j] \in P'_2, \\ \mathtt{oi}, \qquad\qquad \text{otherwise,} \end{cases}$$

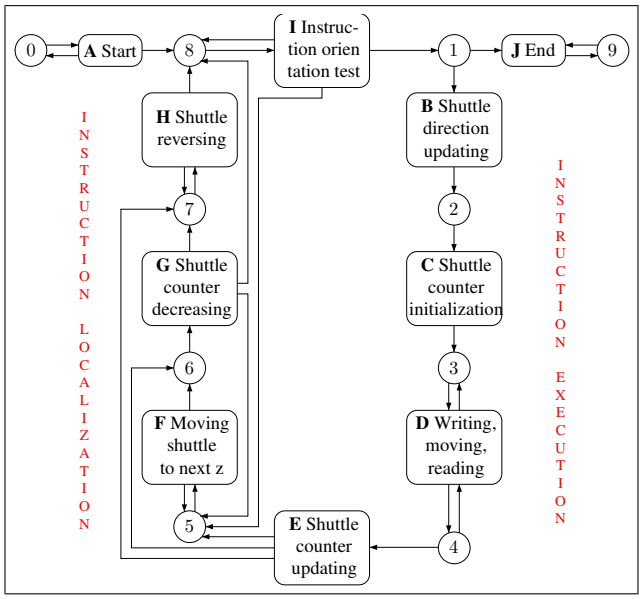
with,

$$\overline{[q_i, a, b, s, q_j]} = \begin{cases} \mathtt{i}a_m \ldots a_2\mathtt{o}, \text{ if } \pi(j) - h(i,a) > 0, \\ \mathtt{o}a_2 \ldots a_m\mathtt{i}, \text{ if } \pi(j) - h(i,a) < 0, \end{cases}$$

with a_2a_3 equal to $\mathtt{io}$, $\mathtt{oi}$, $\mathtt{ii}$, depending whether b equals $\mathtt{u}$, $\mathtt{o}$, $\mathtt{i}$, $\mathtt{z}$, with $a_4 = \mathtt{o}$ or $a_4 = \mathtt{i}$ depending whether $s = +$ or $s = -$ and with $\mathtt{i}a_m \ldots a_5$ a binary number ($\mathtt{o}$ for 0 and $\mathtt{i}$ for 1) whose value is equal to $|\pi(j) - \pi(i,a)| + \frac{3}{2}$.

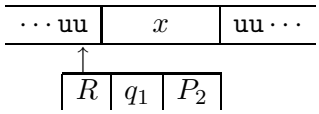
5.3 Operation of the universal pair $(U_2, code_2)$

As already mentioned, the program U_2 has 54 states, $q_1, \ldots, q_{54}$, and 184 instructions. These instructions are divided in 10 modules $A, B, C, \ldots, J$ organized as follows:

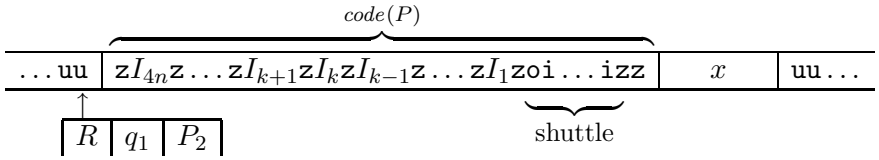


The numbers 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 denote respectively the states q_1, q_{24}, q_{35}, q_{43}, q_{49}, q_{15}, q_{13}, q_7, q_{10}, q_{23}. The complete program U_2 is given in the appendix as a graph whose vertices are the states and the edges the instructions of U_2: each instruction $[q_i, abs, q_j]$ is represented by an arrow, labeled abs, going from q_i to q_j. Note that the vertices a, b, c and 7 have two occurrences which must be merged.

Initial configurations Initially the machines executing P_2 is in the configuration



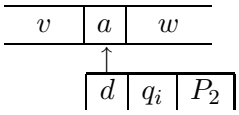
and the machine executing u_2 is in the *correponding initial configuration*



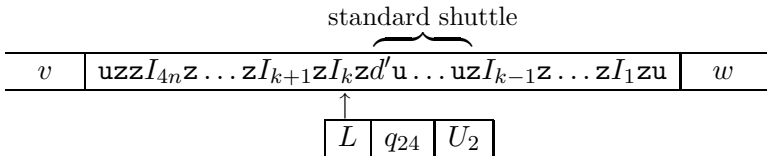
While the machine executing P_2 performs no transitions, the machine executing U_2 performs a sequence of initial transitions, always the same, involving the

instructions of module A and some instructions already there in the modules I, H, G, F. Then the machines executing P_2 and U_2 end up respectively in the following current configurations with $k = 1$:

Current configurations While the machine excuting P_2 is in the current configuration

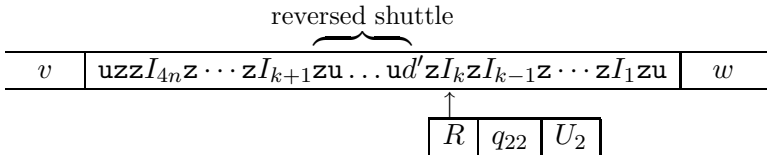


the machine executing U_2 is in the *corresponding current configuration*



(5)

or



(6)

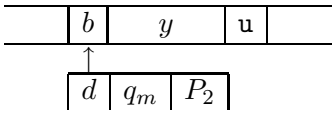
depending whether I_k, with $k = \pi(i, a)$, is in the *standard* form $\mathtt{i}a_m \ldots a_2\mathtt{o}$ or in the *reversed* form $\mathtt{i}a_m \ldots a_2\mathtt{o}$. The read-write points to a_3 or to the $\mathtt{z}$ which follows I_k when I_k is the empty instruction $\mathtt{oi}$. Depending whether d is equal to L or R, the symbol d' is equal to $\mathtt{u}$ or $\mathtt{o}$, if I_k is standard, and to $\mathtt{o}$ or $\mathtt{u}$, if I_k is reversed.

While the current configuration of P_2 is not final, P_2 performs one transition for reaching the next current configuration and U_2 performs a sequence of transitions for reaching the next corresponding current configuration. More precisely, using the information contained in I_k, the program U_2

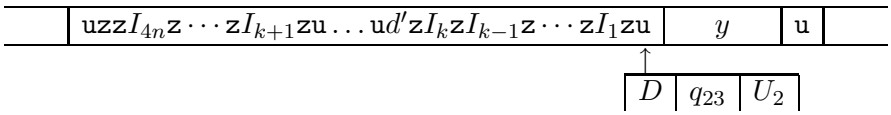
- updates the internal direction contained in the shuttle (module B),
- transfers in the shuttle the binary number serving as basis for computing the number of instructions to be jumped toward the left or the right, depending on whether the shuttle is standard or reversed (module C),
- simulates the writing of a symbol, the read-write head move, and then the reading of a new symbol (module D),
- taking into account the read symbol, updates the binary number contained in the shuttle in order to obtain the right number of instructions to be jumped by the shuttle for reaching the next instruction to be executed (module E),
- moves the shuttle and eventually reverses it, for correctly positioning it alongside the next instruction to be executed (modules F, G, H, I).

When the current configuration of P_2 becomes final, the corresponding current configuration of U_2 is of the form (6) with I_k equal to the empty instruction `oi`. Then U_2 performs a sequence of transitions (module J) for reaching the final corresponding configurations. The machines executing P_2 and U_2 end up respectively in the following final configurations:

Final configurations While the machine executing P_2 terminates in the final configuration



the machine executing U_2 terminates in the *corresponding final configuration*



with $I_k = \texttt{oi}$, $k = \pi(m, b)$ and d' equal to `o` or `u`, depending whether d equals L or R.

6 Complexity and introspection coefficient of our two pairs

6.1 General complexity

Let P_2 be any program for $\mathcal{M}_2$. From the way the coding function $code_2$ is defined, for $\ell = 2$,

$$\boxed{|code_\ell(P_\ell)| = \mathcal{O}(n \log n)},$$

where n is the number of states of $clean(P_\ell)$. This result also holds for $\ell = 1$, with P_1 being any program for $\mathcal{M}_1$, the classical Turing mchine with same alphabet as $\mathcal{M}_2$. This is due to the fact that $code_1(P_1) = code_2(f_1(P_1))$ and that the number of instructions of $clean(f_1(P_1))$ is at most equal to twice the number of instructions of P_1.

Returning to the way U_2 operates at section 5.3, we conclude that if P_2 performs h transitions then there exists positive integers k_i indepent of h or n such that U_2 performs at most:

- $k_1 \log^2 n$ transitions for reaching the first configuration corresponding to the initial configuration of P_2,
- $hk_1 \log^2 n$ transitions for transferring information from an instruction I_i to the adjacent shuttle,
- $hk_2 n \log n$ transitions for simulating the writing of a symbol, the move of the head and the reading of a symbol,

– $hk_3 n \log^2 n$ transitions for moving the shuttle,
– $hk_4 n \log^2 n$ transitions for reaching a final configuration from a configuration corresponding to a final configuration of P_2,

that is all together, at most $hk_5 n \log^2 n$ transitions.

Thus for $\ell = 2$, there exists a positive real number k, independent of $x \in \Sigma^\star$, such that

$$\boxed{\lambda_\ell(P_\ell, x) \leq n \log^2 n}$$

If instead of measuring the complexity in terms of n we do it in terms of $m = |code_\ell(P_\ell)|$, we conclude that there exists a positive real number k, independent of $x \in \Sigma^\star$, such that

$$\boxed{\lambda_\ell(P_\ell, x) \leq m \log m}$$

These two results also hold also for $\ell = 1$, with P_1 being any program for $\mathcal{M}_1$.

6.2 Complexity on examples

On particular examples we have obtained the following complexity results for the pairs $(U_\ell, code_\ell)$, with $\ell = 1$ and $\ell = 2$,

x	$cost(P_\ell, x)$	$cost(U_\ell, code_\ell(P_\ell)\cdot x)$	$cost(U_\ell, code_\ell(U_\ell)\cdot code_\ell(P_\ell)\cdot x)$	$\lambda_\ell(P_\ell, x)$	$\lambda_\ell(U_\ell, code_\ell(P_\ell)\cdot x)$
ε	2	5 927	22 974 203	2 963.50	3 876.19
`o`	6	13 335	51 436 123	2 222.50	3 857.23
`oi`	12	23 095	88 887 191	1 924.58	3 848.76
`oiz`	20	35 377	136 067 693	1 768.85	3 846.22
`oizo`	30	49 663	190 667 285	1 655.43	3 839.22

Here P_ℓ, with $P_1 = f_2(P_2)$, is a reversing program such that, for all $n \geq$, one gets $out(P_\ell, a_1 a_2 \dots a_n) = a_n \dots a_2 a_1$, with the a_i's taken from $\{\mathtt{o}, \mathtt{i}, \mathtt{z}\}$. In both cases, $\ell = 1$ and $\ell = 2$, the program P_ℓ has 32 instructions and 9 states. We have $|code_\ell(P_\ell)| = 265$ and $|code_\ell(U_\ell)| = 1552$.

6.3 Introspection coefficient

To satisfy Hypothesis 2, $code_2(U_2) = f_1(f_2(code_2(U_2)))$ and the labelling function μ is defined so that, for all transitions (c_1, c_2) and (c'_1, c'_2) of $\bigcup U_2$ the integers $\mu(c_1, c_2)$ and $\mu(c'_1, c_2\prime)$ are equal if and only if all the conditions below are satisfied:

– the states of c_1 and c'_1 are equal,
– the symbols pointed by the read-write heads in c_1 and c'_1 are equal,
– the symbols pointed by the read-write heads in c_2 and c'_2 are equal,
– the directions in c_2 and c'_2 are the same.

The function Φ is defined, for all configuration of U_2, with $P_2 = U_2$ by,

$$\Phi(c) = \begin{cases} \text{current configuration corresponding to } c, \text{ if } c \text{ is not final for } P_2, \\ \text{final configuration corresponding to } c, \text{ if } c \text{ is final for } P_2. \end{cases}$$

After having computed the column vector B and the matrix A, using Theorem 2, we have verified that U_2 admits an introspect coefficient and computed its value: for $\ell = 2$ and all words x on Σ such that $cost(P,\, x) \neq \infty$,

$$\boxed{\lim_{n\to\infty} \frac{cost(U_\ell,\, code_\ell(U_\ell)^{n+1}\cdot x)}{cost(U_\ell,\, code(U_\ell)^n \cdot x)} = 3\,672.98}$$

This result also holds for $\ell = 1$.

7 Our universal program for the indirect addressing arithmetic machine

It is interesting to compare the complexities of our universal program for a Turing machine with the complexity of a universal program for the indirect addressing arithmetic machine with same alphabet $\Sigma = \{c_1, c_2, c_3\}$, with $c_1 =$ `o`, $c_2 =$ `i` and $c_3 =$ `z`.

We have written such a universal program U_3 using 103 instructions. From the operation of our universal pair $(U_3, code_3)$ we have been able to show that:

Property 6 *There exists a positive number k such that, for all programs P_3 and word x on Σ, with $cost(P_3,\, x) \neq \infty$,*

$$\boxed{\lambda_3(P_3,\, x) = \frac{cost(U_3,\, code_3(P_3)\cdot x)}{cost(P_3,\, x)} \leq 35 + k\frac{|code(P)|}{cost(P_3,\, x)}}$$

Thus the size of the program P_3 becomes irrelevant when a large number of transitions is performed. On particular examples we have obtained the following results:

x	$cost(P_3, x)$	$cost(U_3, code_3(P_3)\cdot x)$	$cost(U_3, code_3(U_3)\cdot code_3(P_3)\cdot x)$	$\lambda_3(P_3, x)$	$\lambda_3(U_3, code_3(P_3)\cdot x)$
ε	12	2 372	72 110	197, 66	30, 40
`o`	16	2 473	74 758	154, 56	30, 23
`oi`	31	2 860	84 916	92, 26	29, 69
`oiz`	35	2 961	87 564	84, 60	29, 57
`oizo`	50	3 348	97 722	66, 96	29, 19

where P_3 is a reversing program of 21 instructions such that, for all $n \geq 0$ one obtains $out(P,\, a_1a_2\ldots a_n) = a_n\ldots a_2a_1$, with the a_i's taken from $\{$`o`, `i`, `z`$\}$. Additionally, $|code_3(P_3)| = 216$ and $|code_3(U_3)| = 1042$.

The introspection coefficient obtained is:

$$\boxed{\lim_{n\to\infty} \frac{cost(U_3,\, code(U_3)^{n+1}\cdot x)}{cost(U_3,\, code(U_3)^n \cdot x)} = 26.27}$$

8 Conclusion

Unless one "cheats", it is difficult to improve the introspection coefficient of our universal Turing machine which took us considerable development effort. Suppose, which is the case, that we have at our disposal a first universal pair $(U, code)$ for a Turing machine.

A first way of cheating consists of constructing the pair $(U, code')$ from the universal pair $(U, code)$, with

$$code'(P) = \begin{cases} \varepsilon, & \text{if } P = U, \\ code(P), & \text{if } P \neq U. \end{cases}$$

Then we have

$$\frac{cost(U', code(U')^{n+1} \cdot x)}{cost(U', code(U')^{n} \cdot x)} = \frac{cost(U', x)}{cost(U', x)} = 1$$

and $(U, code')$ is a universal pair with an introspection coefficient equal to 1.

There is a second more sophisticated way of cheating, without modifying the coding function *code*. Starting from the universal program U we construct a program U', which, after having erased as many times as possible a given word z occurring as prefix of the input, behave as U on the remaining input. According to the recursion theorem [2, 4], it is possible to take z equal to $code(U')$ and thus to obtain a universal program U' such that, for all $y \in \Sigma^\star$ having not $code(U)'$ as prefix,

$$cost(U',\ code(U')^n \cdot y) = nk_1 + k_2(y),$$

where k_1 and $k_2(y)$ are positive integers, with k_1 being independent of y. Then we have

$$\frac{cost(U', code(U')^{n+1} \cdot y)}{cost(U', code(U')^{n} \cdot y)} = \frac{cost(U, x) + (n+1)k_1 + k_2(y)}{cost(U, x) + nk_1 + k_2(y)} =$$

$$1 + \frac{k_1}{cost(U, x) + k_2(y) + nk_1}.$$

By letting n tend toward infinity we obtain an introspection coefficient equal to 1 for the pair $(U', code)$.

Unfortunately our introspection coefficient definition, page 8, does not disallow these two kinds of cheating. By imposing Hypothesis 2, the first way of cheating is still possible and the second one can be prevented by imposing that the function φ never produces the empty sequence. But this last restriction seems to be *ad hoc*.

What one really would like to prevent is that the function *code* or the program U "behaves differently" on the program P, depending whether P is or is not equal to U. It is an open problem to express this restriction in the definition of the introspection coefficient.

References

1. Marvin Minsky, *Computations: Finite and Infinite Machines,* Prentice-Hall, 1967.
2. Hartley Rogers, *Theory of Recursive Functions and Effective Computability,* McGraw-Hill, 1967, also MIT Press, fifth printing, 2002.

3. Yurii Rogozin, Small universal Turing machines, *Theoretical Computer Science,* Volume 168, number 2, november 1996.
4. Michael Sipser, *Introduction to the Theory of Computation,* PWS Publishing Company, 1997.

Appendix: graph of the universal program U_2