Back to the Complexity of Universal Programs

Alain Colmerauer

Marseilles, France

Abstract. I start with three examples illustrating my contribution to constraint programming: the problem of cutting a rectangle into different squares in Prolog III, a complicated constraint for Prolog IV, the optimal narrowing of the sortedness constraint. Then I switch to something quite different: to machines, in particular to Turing machines. After the declarative aspect, the basic computational aspect!

The 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 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.

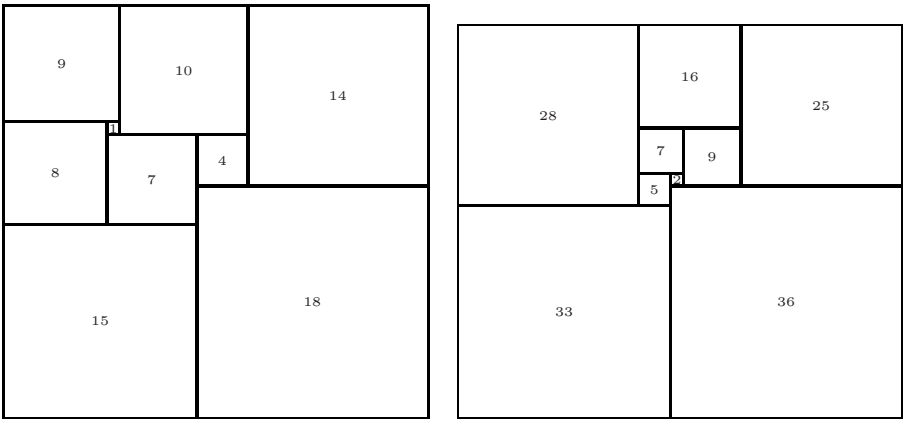
1 Preface

Let us review my contribution to constraint programming.

1.1 Around 1985

Around 1985 I was interested by constraints, more precisely by numerical linear constraints, by Boolean algebra and by list constraints. That's how Prolog III was born [2]. A good example of a program consists in cutting a rectangle of unknown size into n different squares also of unknown sizes. For $n = 9$ the following result holds:

P.J. Stuckey (Ed.): CP 2008, LNCS 5202, pp. 1–20, 2008.



Here is the program, written in the syntax of Prolog IV. The height of the rectangle to be cut is assumed to be 1, which is not a restriction:

```
rectangle(A,C) :-
   gelin(A,1),
   distinctSizes(C),
   area([-1,A,1],L,C,[]).

distinctSizes([]).
distinctSizes([B|C]) :-
   gtlin(B,0),
   distinctSizes(C),
   out(B,C).

out(B,[]).
out(B,[Bp|C]) :-
   dif(B,Bp),
   out(B,C).

area([V|L],[V|L],C,C) :-
   gelin(V,0).
area([V|L],Lppp,[B|C],Cpp) :-
   lt(V,0),
   square(B,L,Lp),
   area(Lp,Lpp,C,Cp),
   area([V+B,B|Lpp],Lppp,Cp,Cpp).

square(B,[H,0,Hp|L],Lp) :-
   gtlin(B,H),
   square(B,[H+Hp|L],Lp).
square(B,[H,V|L],[-B+V|L]) :-
   B = H.
square(B,[H|L],[-B,H-B|L]) :-
   ltlin(B,H).
```

The predicates $gelin(x,y)$, $gtlin(x,y)$, $ltlin(x,y)$ correspond to the linear constraints $x \geq y$, $x > y$, $x < y$ and $dif(x,y)$ to the constraint $x \neq y$. We leave the program uncommented. It is sufficient to ask the query

```
>> size(C)=9, rectangle(A,C).
```

where $size(x) = y$ means *size of the list* x *is* y, to obtain

```
A = 33/32,
C = [15/32,9/16,1/4,7/32,1/8,7/16,1/32,5/16,9/32];
A = 69/61,
C = [33/61,36/61,28/61,5/61,2/61,9/61,25/61,7/61,16/61];
A = 33/32,
C = [9/16,15/32,7/32,1/4,7/16,1/8,5/16,1/32,9/32];
A = 69/61,
C = [36/61,33/61,5/61,28/61,25/61,9/61,2/61,7/61,16/61];
```

```
A = 33/32,
C = [9/32,5/16,7/16,1/4,1/32,7/32,1/8,9/16,15/32];
A = 69/61,
C = [28/61,16/61,25/61,7/61,9/61,5/61,2/61,36/61,33/61];

A = 69/61,
C = [25/61,16/61,28/61,9/61,7/61,2/61,5/61,36/61,33/61];
A = 33/32,
C = [7/16,5/16,9/32,1/32,1/4,1/8,7/32,9/16,15/32].
```

1.2 Around 1990

Prolog IV was finished in 1995 [3]. In addition to the constraints of Prolog III, it includes numerical non-linear constraints which are approximately solved by narrowing of intervals. It also includes the existential quantifier. Here, on the left column, is a constraint in the usual notation and the value of the free variable y. The formula $(x > y)$ denotes the Boolean value *true* ou *false*. On the right column you find the corresponding query and the answer in Prolog IV.

$$\exists u \exists v \exists w \exists x \begin{pmatrix} y \leq 5 \\ \wedge\ v_1 = \cos v_4 \\ \wedge\ \text{size}(u) = 3 \\ \wedge\ \text{size}(v) = 10 \\ \wedge\ u \bullet v = v \bullet w \\ \wedge\ y \geq 2 + (3 \times x) \\ \wedge\ x = (74 > \lfloor 100 \times v_1 \rfloor) \end{pmatrix}$$

$$y = 5$$

```
>> U ex V ex W ex X ex
le(Y,5),
V:1 = cos(V:4),
size(U) = 3,
size(V) = 10,
U o V = V o W,
ge(Y,2.+.(3.*.X)),
X = bgt(74,floor(100.*.V:1)).

Y = 5.
```

1.3 Around 2000

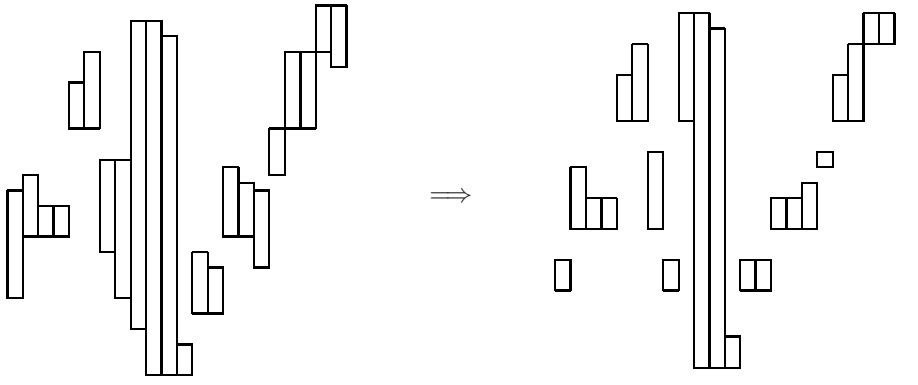
Having been interested by the narrowing of intervals, I focused on a particular instance, the sortedness constraint:

$$sort(x_1, \ldots, x_n, x_{n+1}, \ldots, x_{2n}) \quad \equiv \quad \begin{cases} (x_{n+1}, \ldots, x_{2n}) \\ \text{is equal to} \\ (x_1, \ldots, x_n) \text{ sorted} \\ \text{in non-decreasing order.} \end{cases}$$

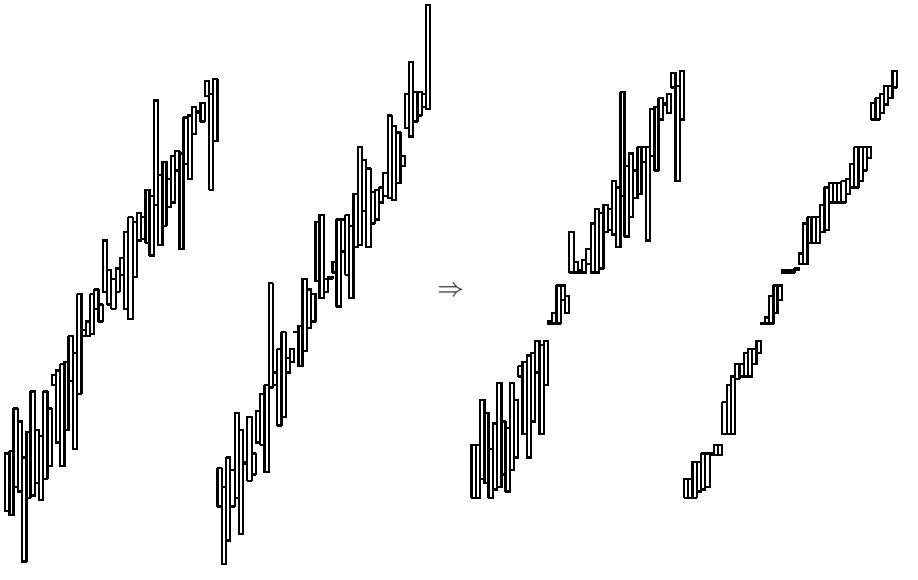
With Noëlle Bleuzen [1], a colleague from the department of Mathematics, we developed an algorithm of complexity $O(n \log n)$ to compute the smallest intervals a'_i from the intervals a_i such that:

$$\begin{gathered} sort(x_1, \ldots, x_{2n}) \wedge x_1 \in a_1 \wedge \cdots \wedge x_{2n} \in a_{2n} \\ \equiv \\ sort(x_1, \ldots, x_{2n}) \wedge x_1 \in a'_1 \wedge \cdots \wedge x_{2n} \in a'_{2n} \end{gathered}$$

For $2n = 22$ for example, we obtain:



and for $2n = 100$:



2 Introduction

In parallel, around 2000, I was 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 a reasonable time. To simulate the execution of one of its own instructions, the last program executes an average number of 3 672.98 instructions. That is the *introspection coefficient,* a key concept of this paper.

The rest of this paper presents this result in a more general context concerning machines other than Turing machines. Section 2 is this introduction. In Section 3, we formally define the concepts of programmed machine, machine, program, transition and instruction. We illustrate this on a Turing machine, and on an indirect addressing arithmetic machine. In Section 4, we introduce the universal pair, program and coding function. We mention the theorem on how to check the existence of its introspection coefficients and how to compute its value. We omit the proof and refer the reader to [4]. Sections 5 and 6 and also Appendix 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. In Section 7 we conclude about a lack of restriction of our definition of the introspection coefficient.

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 [5] and Y. Rogozin [7] in the design of universal programs for Turing machines with very small numbers of states. Surprisingly, they seem particularly inefficient in terms of the number of executed instructions.

3 Machines

3.1 Basic Definitions

Definition 1. *A* machine M *is a 5-tuple* $(\Sigma, C, \alpha, \omega, I)$*, where*

Σ, *the alphabet of* 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* $\Sigma^\star$*;*
I, *is a countable set of* instructions*, an intruction being a set of* compatibles *transitions, i.e., whose first components are all distinct.*

Definition 2. *A* program P *for a machine* M *is a finite subset of the instructions set* I *of* M*, such that the transitions of* $\bigcup P$ *are compatible.*[1]

3.2 How a Machine Operates

Let $M = (\Sigma, C, \alpha, \omega, I)$ be a machine and P a program for M. The operation of the machine (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:

$$orbit_M(P,\, x) = \begin{cases} \text{the longest sequence } (c_0, c_1)\,(c_1, c_2)\,(c_2, c_3)\,\ldots \text{ with} \\ c_0 = \alpha(x) \text{ and each } (c_i, c_{i+1}) \text{ an element of } \bigcup P. \end{cases}$$

$$out_M(P,\, x) \;\; = \begin{cases} \nearrow, & \text{if } orbit(P,\, x) \text{ is infinite,} \\ \omega(c_n), & \text{if } orbit(P,\, x) \text{ ends with } (c_{n-1}, c_n) \end{cases}$$

3.3 Example: 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 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.

In fact we consider a variant of the Turing machines described above 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.

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 3. *A* Turing *machine is a 5-tuple of the form* $(\Sigma, C, \alpha, \omega, I)$ *where,*

- Σ *is a finite set not having* $\mathtt{u}$ *as an element,*
- 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* $\Sigma^\star_{\mathtt{u}}$ *and* a *taken from* $\Sigma_{\mathtt{u}}$*, where* $\Sigma_{\mathtt{u}} = \Sigma \cup \{\mathtt{u}\}$*,*
- $\alpha(x) = [R, q_1, \varepsilon, \mathtt{u}, x]$*, for all* $x \in \Sigma^\star$*,*
- $\omega([d, q_i, \cdot x, a, y\cdot])$ *is the longest element of* $\Sigma^\star$ *beginning* $y\cdot$*,*
- 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*
  $[q_i,\, abs,\, q_j] \stackrel{def}{=}$
  $\{([d, q_i, \cdot xc, a, y\cdot],\, [L, q_j, \cdot x, c, by\cdot]) \mid\ (d, s) \in E_1 \text{ and } (x, c, y) \in F\} \cup$
  $\{([d, q_i, \cdot x, a, cy\cdot],\, [R, q_j, \cdot xb, c, y\cdot]) \mid\ (d, s) \in E_2 \text{ and } (x, c, y) \in F\},$
  with $E_1 = \{(L, +), (R, -)\}$*,* $E_2 = \{(R, +), (L, -)\}$ *and* $F = \Sigma^\star_{\mathtt{u}} \times \Sigma_{\mathtt{u}} \times \Sigma^\star_{\mathtt{u}}$*.*

[2] Index M is omitted when there is no ambiguity.

3.4 Example: 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 4. *An* indirect addressing arithmetic *machine is a 5-tuple of the form* $(\Sigma, C, \alpha, \omega, I)$*, where,*

- $\Sigma = \{c_1, \ldots, c_m\}$*, where the* c_i *are any symbols,*
- 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* 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* $\{1, \ldots, m\}$*,*
- I *is the set of instructions denoted and defined, for all natural integers* i, j, k*, by:*

$$[i, cst, j, k] \stackrel{def}{=} \{(r,\, s) \in C^2 \,|\, r_0 = i, s := r,\; s_j := k,\; s_0 := s_0 + 1\},$$

$$[i, plus, j, k] \stackrel{def}{=} \{(r,\, s) \in C^2 \,|\, r_0 = i, s := r,\; s_j := s_j + s_k,\; s_0 := s_0 + 1\},$$

$$[i, sub, j, k] \stackrel{def}{=} \{(r,\, t) \in C^2 \,|\, r_0 = i,\; s := r,\; s_j := s_j \div s_k,\; s_0 := s_0 + 1\},$$

$$[i, from, j, k] \stackrel{def}{=} \{(r,\, t) \in C^2 \,|\, r_0 = i,\; s := r,\; s_j := s_{s_k},\; s_0 := s_0 + 1\},$$

$$[i, to, j, k] \stackrel{def}{=} \{(r,\, t) \in C^2 \,|\, r_0 = i,\; s := r,\; s_{r_j} = r_k,\; s_0 := s_0 + 1\},$$

$$[i, ifze, j, k] \stackrel{def}{=} \{(r,\, t) \in C^2 \,|\, r_0 = i,\; s := r,\; s_0 := \begin{bmatrix} s_k + 1, \text{ if } s_j = 0 \\ s_0 + 1, \text{ if } s_j \neq 0 \end{bmatrix}\}.$$

Here $s_j \div s_j$ *stands for* $\max\{0, s_j - s_k\}$.

4 Universal Program and Universal Coding

4.1 Universal Pair

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

Definition 5. *The pair* $(U,\, code)$*, the program* U *and the coding function code, are said to be* universal *for* M*, if, for all programs* P *of* M *and for all* $x \in \Sigma^\star$*,*

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

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

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

4.2 Introspection Coefficient

Let $(U, code)$ be a universal pair for the machine $M = (\Sigma, C, \alpha, \omega, I)$. 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 6. *Given a program* P *for* M *and a word* x *on* Σ *with* $orbit(P,\ x) \neq \nearrow$, *the* complexity *of* $(U,\ code)$ *is the real number defined by*

$$\frac{|orbit(U,\ code(P) \cdot x)|}{|orbit(P,\ x)|}.$$

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

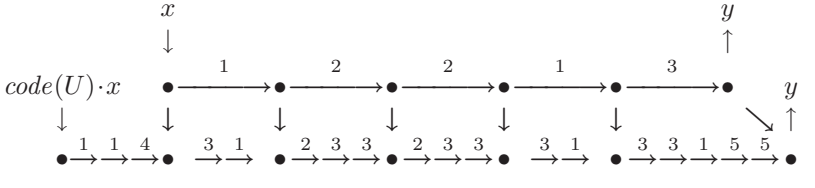
Definition 7. *If for all* $x \in \Sigma^\star$, *with* $orbit(U,\ x) \neq \nearrow$, *the real number*

$$\lim_{n \to \infty} \frac{|orbit(U, code(U)^{n+1} \cdot x)|}{|orbit(U, code(U)^n \cdot x)|}$$

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

4.3 Existence and Value of the Introspection Coefficient

Let $(U, code)$ be a universal pair for a machine $M = (\Sigma, C, \alpha, \omega, I)$. 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 1. *There exists $n, nb, \mathcal{A}, \mathcal{B}$ such that, for every pair of traces of the form*

$$(orbit(U,\ code(U)\cdot x,\ orbit(U,\ x)))$$

itself of the form

$$(s_1 \cdots s_l,\ r_1 \cdots r_k),$$

we have

$$nb(s_1) \cdots nb(s_l) = \mathcal{B} \cdot \mathcal{A}(nb(r_1)) \cdots \mathcal{A}(nb(r_k)),$$

with n positive integer, with $nb(t) \in 1..n$ for each transition t of U, with $\mathcal{A}(i)$ a finite sequence on $1..n$ for each $i \in 1..n$, with $\mathcal{B}$ a finite sequence on $1..n$.

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

$$B = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}, \qquad b_i = \text{number of occurrences of integer } i \text{ in } \mathcal{B},$$
$$A = \begin{bmatrix} a_{11} \cdots a_{nn} \\ \vdots \qquad \vdots \\ a_{1n} \cdots a_{nn} \end{bmatrix}, \ a_{ij} = \text{number of occurrences of integer } i \text{ in } \mathcal{A}(j). \tag{3}$$

and we conclude by the theorem, where $||X||$ denotes the sum of the components of X:

Theorem 1. *Suppose the matrix A admits a real eigenvalue λ, whose multiplicity is equal to 1, which is strictly greater to 1 and to the greatest modulus λ' of the other eigenvalue.*

If α is a real number with $\lambda' < \alpha < \lambda$, if $X_0 = B$ and $X_{n+1} = \frac{1}{\alpha} A X_n$, then, when $n \to \infty$, exactly one of the two properties holds:

1. $||X_n|| \to 0$,
2. $||X_n|| \to \infty$. *In this case λ is the inspection coefficient.*

Anyone interested in more details may consult [4].

5 Universal Pair for the Turing Machine

5.1 The Universal Pair

We now present a particularly efficient universal pair $(U, code)$ for the Turing machine M with alphabet $\Sigma = \{\mathtt{o}, \mathtt{i}, \mathtt{z}\}$. The program U has 184 instructions and 54 states and $|code(U)| = 1552$. Its listing and its graph can be seen in the annexes A and B.

5.2 Coding Function of the Universal Pair

Let P be a program for M. We take $code(P)$ as the word on $\{\mathtt{o}, \mathtt{i}, \mathtt{z}\}$

$$code(P) = \mathtt{z}I_{4n}\mathtt{z} \ldots \mathtt{z}I_{k+1}\mathtt{z}I_k\mathtt{z}I_{k-1}\mathtt{z} \ldots \mathtt{z}I_1\mathtt{zoi} \ldots \mathtt{izz}.$$

Integer n is the number of states of P and the I_k are the coded instructions. The size of the *shuttle* $\texttt{oi}\ldots\texttt{iz}$ is equal to the longest size of the I_k minus 5.

In order to assign a position to each coded instruction I_k of $[q_i, abs, q_j]$, we introduce the number:

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

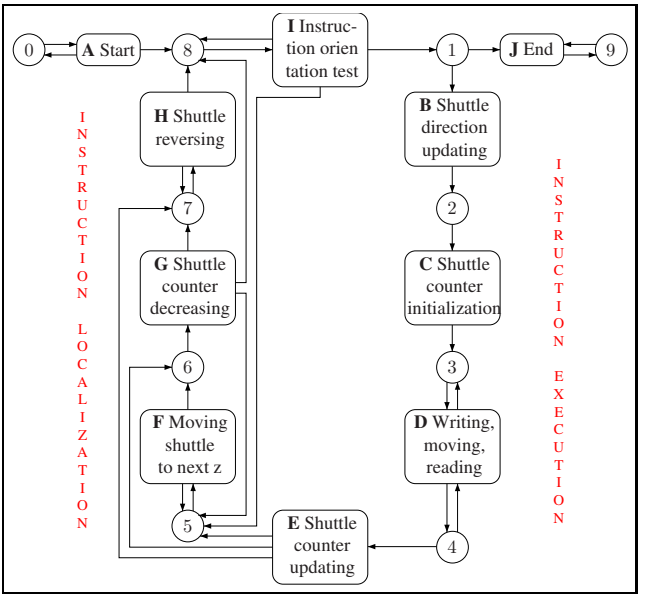
For all $a \in \Sigma_{\texttt{u}}$ and $i \in 1..n$,

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

- with $\overline{[q_i, a, b, s, q_j]} = \begin{cases} \texttt{i}a_m \ldots a_2\texttt{o}, \text{ if } \pi(i,a) < \frac{1}{2}(\pi(j,\texttt{u}) + \pi(j,\texttt{z})), \\ \texttt{o}a_2 \ldots a_m\texttt{i}, \text{ if } \pi(i,a) > \frac{1}{2}(\pi(j,\texttt{u}) + \pi(j,\texttt{z})), \end{cases}$
- with a_2a_3 equal to $\texttt{io}$, $\texttt{oi}$, $\texttt{ii}$, depending whether b equals $\texttt{u}$, $\texttt{o}$, $\texttt{i}$, $\texttt{z}$,
- with $a_4 = \texttt{o}$ or $a_4 = \texttt{i}$ depending whether $s = +$ or $s = -$ and
- with $\texttt{i}a_m \ldots a_5$ a binary number ($\texttt{o}$ for 0 and $\texttt{i}$ for 1) whose value is equal to $|\pi(j) - \pi(i,a)| + \frac{3}{2}$.

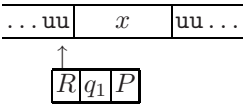
5.3 Operation of the Universal Pair

As already mentioned, the program U 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:

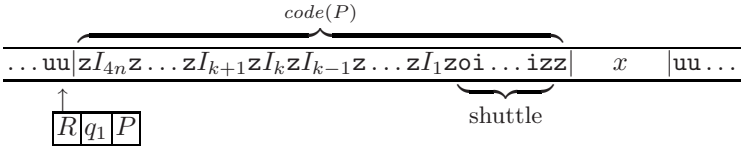


The numbers 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 denote respectively the states q_1, q_{24}, q_{35}, q_{43}, q_{49}, q_{15}, q_{13}, q_7, q_{10}, q_{23}. They are called $X0$, $X2$, ..., $X9$ in the program U in the annex A. In the annex B, we also give a graph whose vertices are the states and the edges the instructions of U: 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 is in the configuration

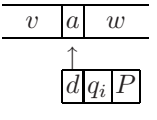


and the machine executing U is in the *correponding initial configuration*

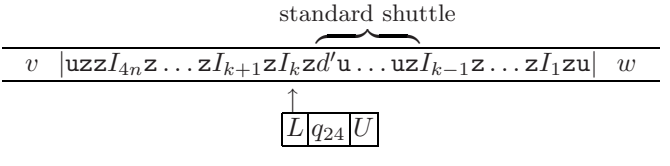


While the machine executing P performs no transitions, the machine executing U performs a sequence of initial transitions, always the same, involving the instructions of module A and some instructions already there in the modules I, H, G, F. Then the machines executing P and U end up respectively in the following current configurations with $k = 1$:

Current configurations. While the machine executing P is in the current configuration

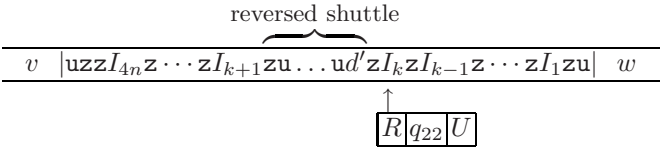


the machine executing U is in the *corresponding current configuration*



(4)

or



(5)

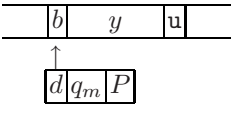
depending whether I_k, with $k = \pi(i, a)$, is in the *standard* form $\mathtt{i}a_m \dots a_2\mathtt{o}$ or in the *reversed* form $\mathtt{i}a_m \dots a_2\mathtt{o}$. The read-write points to a_3 or to the `z` which follows I_k when I_k is the empty instruction `oi`. Depending whether d is equal to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

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

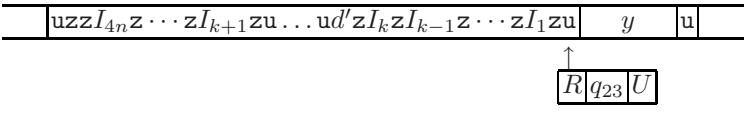
- Updates the internal direction contained in the shuttle (module B),
- Transfers in the shuttle the binary number serving as basis for computing the number of instructions to be jumped toward the left or the right, depending on whether the shuttle is standard or reversed (module C),
- Simulates the 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 becomes final, the corresponding current configuration of U is of the form (5) with I_k equal to the empty instruction `oi`. Then U performs a sequence of transitions (module J) for reaching the final corresponding configurations. The machines executing P and U end up respectively in the following final configurations:

Final configurations. While the machine executing P terminates in the final configuration



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



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

5.4 Introspection Coefficient of Our Pair for the Turing Machine

First we have chosen a reversing program P such that, for all $n \geq$, one gets $out(P, a_1a_2 \ldots a_n) = a_n \ldots a_2a_1$, with the a_i taken from $\{\mathtt{o}, \mathtt{i}, \mathtt{z}\}$. The program P has 32 instructions and 9 states. We have $|code(P)| = 265$ and $|code(U)| = 1552$. We obtain the following results for the pair $(U, code)$:

x	$\lvert orbit(P, x)\rvert$	$\lvert orbit(U, code(P)\cdot x)\rvert$	$\lvert orbit(U, code(U)\cdot code(P)\cdot x)\rvert$	$\frac{\lvert orbit(U,\, code(U)\cdot code(P)\cdot x)\rvert}{\lvert orbit(U,\, code(P)\cdot x)\rvert}$
ε	2	5 927	22 974 203	3 876.19
o	6	13 335	51 436 123	3 857.23
oi	12	23 095	88 887 191	3 848.76
oiz	20	35 377	136 067 693	3 846.22
oizo	30	49 663	190 667 285	3 839.22

It can be seen that we have succeeded in running the universal program U on its own code and thus to compute a first approximation of the introspection coefficient.

Second, after having computed the column vector B of size $184 \times 4 = 736$ and the matrix A of size 736, using Theorem 1, we have verified that U admits an introspection coefficient and computed its value: for all words x on Σ such that $orbit(P, x) \neq \nearrow$,

$$\lim_{n\to\infty} \frac{|orbit(U,\ code_1(U)^{n+1}\cdot x)|}{|orbit(U,\ code(U)^n\cdot x)|} = 3\,672.98$$

Anyone interested in more details may consult [4]. There, it is also proven that a more classical Turing machine, with 361 instructions and 106 states, has the same introspection coefficient.

6 Universal Pair for the Indirect Addressing Arithmetic Machine

6.1 The Universal Pair

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 $c_1 = \mathtt{o}$, $c_2 = \mathtt{i}$ and $c_3 = \mathtt{z}$. We have written such a universal program U using 103 instructions and with $|code(U)| = 1042$. It can be seen in annex C.

6.2 Operation of the Universal Pair

The universal pair $(U, code)$ for arithmetic machine with indirect addressing operates roughly as following:

Current configuration

r_0	r_1	r_2	
0	3	2	

Execution of one instruction of

$$P = \left\{ \begin{array}{l} [0, \mathit{plus}, 2, 1], \\ [1, \mathit{cst}, 11, 1], \\ [2, \mathit{from}, 5, 2], \\ [3, \mathit{ifze}, 5, 8], \\ [4, \mathit{sub}, 5, 11], \\ [5, \mathit{to}, 2, 5], \\ [6, \mathit{plus}, 2, 11], \\ [7, \mathit{cst}, 0, 1] \end{array} \right\}$$

Next configuration

r_0	r_1	r_2	
1	3	5	

Corresponding configuration

r_0	r_1	r_2					
50			$\mathit{code}(P)$	0	3	2	

Execution of several corresponding instruction of

$$U = \left\{ \begin{array}{l} [0, \mathit{cst}, 8, 0], \\ [1, \mathit{cst}, 10, 2], \\ [2, \mathit{cst}, 11, 11], \\ \ldots \\ [99, \mathit{cst}, 0, 49], \\ [100, \mathit{plus}, 9, 1], \\ [101, \mathit{from}, 9, 9], \\ [102, \mathit{plus}, 1, 9] \end{array} \right\}$$

Corresponding configuration

r_0	r_1	r_2					
50			$\mathit{code}(P)$	1	3	5	

6.3 Introspection Coefficient of Our Pair for the Indirect Addressing Machine

On particular examples we obtain the following results:

x	$\lvert orbit(P, x)\rvert$	$\lvert orbit(U, code(P)\cdot x)\rvert$	$\lvert orbit(U, code(U)\cdot code(P)\cdot x)\rvert$	$\frac{\lvert orbit(U,\, code(U)\cdot code(P)\cdot x)\rvert}{\lvert orbit(U,\, code(P)\cdot x)\rvert}$
ε	12	2 372	72 110	30.40
o	16	2 473	74 758	30.23
oi	31	2 860	84 916	29.69
oiz	35	2 961	87 564	29.57
oizo	50	3 348	97 722	29.19

where P is a reversing program of 21 instructions, with $|\mathit{code}(P)| = 216$, such that, for all $n \geq 0$ one obtains $\mathit{out}(P, a_1 a_2 \ldots a_n) = a_n \ldots a_2 a_1$, with the a_i taken from $\{\mathtt{o}, \mathtt{i}, \mathtt{z}\}$. The introspection coefficient obtained is:

$$\boxed{\lim_{n\to\infty} \frac{|\mathit{orbit}(U,\ \mathit{code}(U)^{n+1}\cdot x)|}{|\mathit{orbit}(U,\ \mathit{code}(U)^{n}\cdot x)|} = 26.27}$$

Anyone interested in more details may consult [4].

7 Conclusion

Unless one "cheats", it is difficult to improve the introspection coefficient of our universal Turing machine which took us a considerable effort to develop.

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

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

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, behaves as U on the remaining input. According to the recursion theorem [6,8], 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,

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

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

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

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

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. 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.

Finally we would like to mention that we tested our universal programs with a package written in MAPLE 8. In each case this package was also used to calculate and manipulate the matrix A and the vectors B. Notably it was used to compute the eigenvalues of A to obtain the introspection coefficient.

References

1. Bleuzen, N., Colmerauer, A.: Optimal Narrowing of a Block of Sortings in Optimal time. Constaints 5(1-2), 85–118 (2000), `http://alain.colmerauer@free.fr`
2. Colmerauer, A.: An Introduction to Prolog III. Communications of the ACM 33(7), 68–90 (1990), `http://alain.colmerauer@free.fr`

3. Colmerauer, A.: Prolog IV (1995), http://alain.colmerauer@free.fr
4. Colmerauer, A.: On the complexity of universal programs. In: Machine, Computations and Universality (Saint-Petersburg 2004). LNCS, pp. 18–35 (2005), http://alain.colmerauer@free.fr
5. Minsky, M.: shape Computations: Finite and Infinite Machines. Prentice-Hall, Englewood Cliffs (1967)
6. Rogers, H.: Theory of Recursive Functions and Effective Computability. McGraw-Hill, New York (1967); fifth printing. MIT Press (2002)
7. Rogozin, Y.: Small universal Turing machines. Theoretical Computer Science 168(2) (November 1996)
8. Sipser, M.: Introduction to the Theory of Computation. PWS Publishing Company (1997)

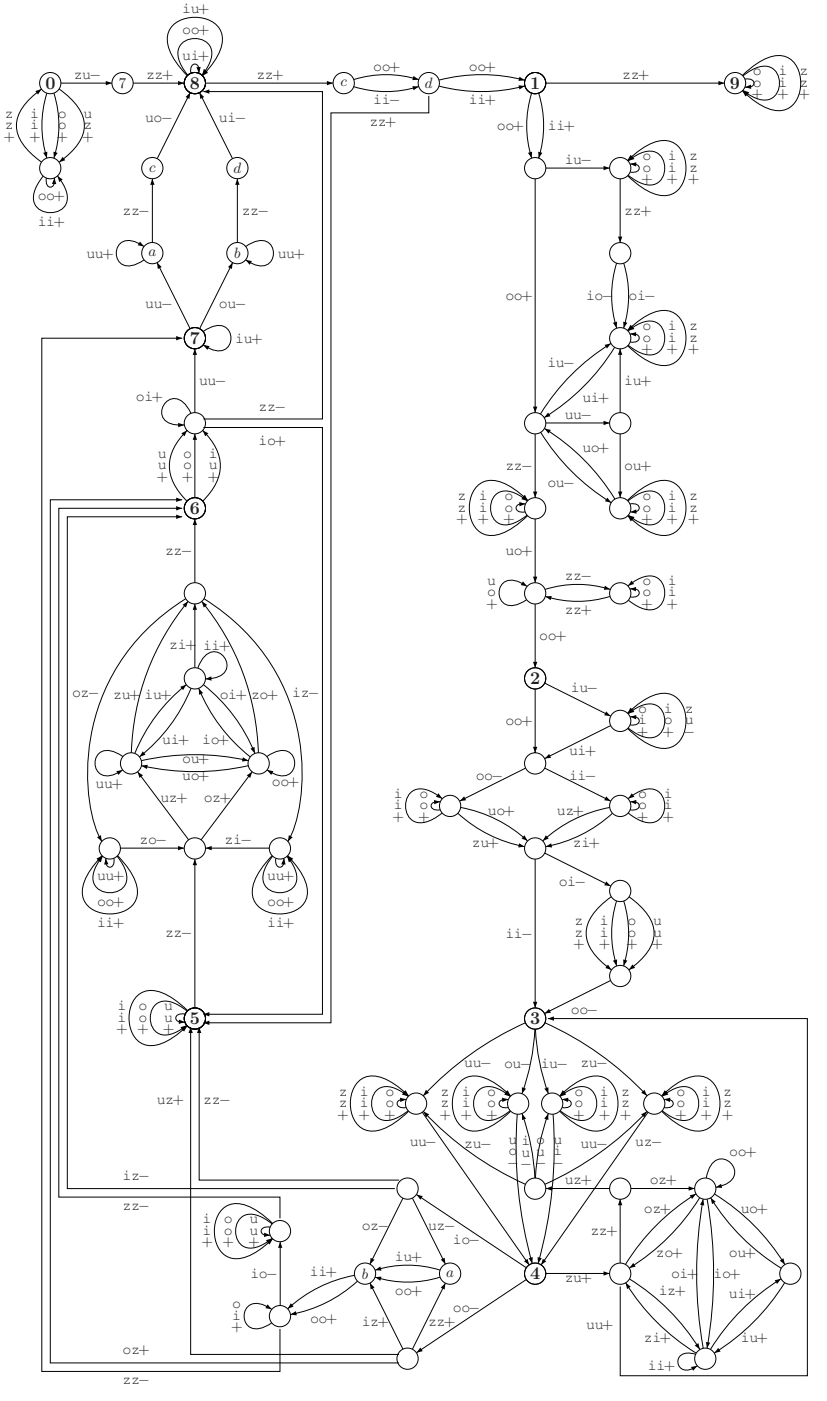
Appendix

A Annex: Universal Turing Program

```
# A BEGINNING
[X0,uz+,A1],   [X0,oo+,A1],   [X0,ii+,A1],   [X0,zu-,X7],
               [A1,oo+,A1],   [A1,ii+,A1],   [A1,zz+,X0],
                                             [X7,zz+,X8],

# B INSTRUCTION TAIL COPYING
               [X1,oo+,B1],   [X1,ii+,B1],
               [B1,oo+,B5],   [B1,iu-,B2],
               [B2,oo+,B2],   [B2,ii+,B2],   [B2,zz+,B3],
               [B3,oi-,B4i],  [B3,io-,B4i],
[B4o,uo+,B5],  [B4o,oo+,B4o], [B4o,ii+,B4o], [B4o,zz+,B4o],
[B4i,ui+,B5],  [B4i,oo+,B4i], [B4i,ii+,B4i], [B4i,zz+,B4i],
[B5,uu-,B6],   [B5,ou-,B4o],  [B5,iu-,B4i],  [B5,zz-,B7],
               [B6,oo+,B4o],  [B6,ii+,B4i],
# Replacement of the remaining u's by o's
[B7,uo+,B9],   [B7,oo+,B7],   [B7,ii+,B7],   [B7,zz+,B7],
[B9,uo+,B9],   [B9,oo+,X2],                  [B9,zz-,B10],
               [B10,oo+,B10], [B10,ii+,B10], [B10,zz+,B9],

# C INSTRUCTION HEAD COPYING
# Creating the symbol to be written
               [X2,oo+,C2],   [X2,iu-,C1],
[C1,ui+,C2],   [C1,oi+,C1],   [C1,io+,C1],   [C1,zu-,C1],
               [C2,oo-,C3o],  [C2,ii-,C3i],
[C3o,uo+,C4],  [C3o,oo+,C3o], [C3o,ii+,C3o], [C3o,zu+,C4],
[C3i,uz+,C4],  [C3i,oo+,C3i], [C3i,ii+,C3i], [C3i,zi+,C4],
# Taking in account the direction
               [C4,oi-,C5],   [C4,ii-,X3],
[C5,uu+,C6],   [C5,oo+,C6],   [C5,ii+,C6],   [C5,zz+,C6],
               [C6,oo-,X3],

# D WRITING, MOVING AND READING
# Writing and reading
[X3,uu-,D1u],  [X3,ou-,D1o],  [X3,iu-,D1i],  [X3,zu-,D1z],
[D0,uu-,D1z],  [D0,ou-,D1i],  [D0,iu-,D1o],  [D0,zu-,D1u],
[D1u,uu-,X4],  [D1u,oo+,D1u], [D1u,ii+,D1u], [D1u,zz+,D1u],
[D1o,uo-,X4],  [D1o,oo+,D1o], [D1o,ii+,D1o], [D1o,zz+,D1o],
[D1i,ui-,X4],  [D1i,oo+,D1i], [D1i,ii+,D1i], [D1i,zz+,D1i],
[D1z,uz-,X4],  [D1z,oo+,D1z], [D1z,ii+,D1z], [D1z,zz+,D1z],
# Moving
                                             [X4,zu+,D2z],
               [D2u,ou+,D2o], [D2u,iu+,D2i],
[D2o,uo+,D2u], [D2o,oo+,D2o], [D2o,io+,D2i], [D2o,zo+,D2z],
[D2i,ui+,D2u], [D2i,oi+,D2o], [D2i,ii+,D2i], [D2i,zi+,D2z],
[D2z,uz+,X3],  [D2z,oz+,D2o], [D2z,iz+,D2i], [D2z,zz+,D2zz],
[D2zz,uz+,D0], [D2zz,oz+,D2o],
```

```
# E SHUTTLE UPDATING
# Beginning of the updating
                [X4,oo-,E1b],  [X4,io-,E1a],
[E1a,uz-,E2a], [E1a,oz-,E2b], [E1a,iz-,X6],  [E1a,zz-,X5],
[E1b,uz+,X5],  [E1b,oz+,X6],  [E1b,iz+,E2b], [E1b,zz+,E2a],
# End of the updating
                [E2a,oo+,E2b], [E2a,iu+,E2b],
                [E2b,oo+,E4],  [E2b,ii+,E4],
                [E4,oi+,E4],   [E4,io-,E5],   [E4,zz-,X7],
[E5,uu+,E5],   [E5,oo+,E5],   [E5,ii+,E5],   [E5,zz-,X6],

# F SUTTLE MOVING TO NEXT z
[X5,uu+,X5],   [X5,oo+,X5],   [X5,ii+,X5],   [X5,zz-,F1],
[F1,uz+,F2u],  [F1,oz+,F2o],
[F2u,uu+,F2u], [F2u,ou+,F2o], [F2u,iu+,F2i], [F2u,zu+,F3],
[F2o,uo+,F2u], [F2o,oo+,F2o], [F2o,io+,F2i], [F2o,zo+,F3],
[F2i,ui+,F2u], [F2i,oi+,F2o], [F2i,ii+,F2i], [F2i,zi+,F3],
                [F3,oz-,F4o],  [F3,iz-,F4i],  [F3,zz-,X6],
[F4o,uu+,F4o], [F4o,oo+,F4o], [F4o,ii+,F4o], [F4o,zo-,F1],
[F4i,uu+,F4i], [F4i,oo+,F4i], [F4i,ii+,F4i], [F4i,zi-,F1],

# G SHUTTLE DECREASING
[X6,uu+,G1],   [X6,oo+,G1],   [X6,iu+,G1],
[G1,uu-,X7],   [G1,oi+,G1],   [G1,io+,X5],   [G1,zz-,X8],

# H SHUTTLE REVERSING AFTER BLANK SYMBOLS INTRODUCTION
[X7,uu-,E2a],  [X7,ou-,E2b],  [X7,iu+,X7],
[E2a,uu+,E2a],                                [E2a,zz-,I1],
[E2b,uu+,E2b],                                [E2b,zz-,I2],
[I1,uo-,X8],
[I2,ui-,X8],

# I INSTRUCTION ORIENTATION TEST AFTER BLANK SYMBOLS INTRODUCTION
[X8,ui+,X8],   [X8,oo+,X8],   [X8,iu+,X8],   [X8,zz+,I1],
                [I1,oo+,I2],   [I1,ii-,I2],
                [I2,oo+,X1],   [I2,ii+,X1],   [I2,zz+,X5],

# J END OF THE PROGRAM
                                              [X1,zz+,X9],
                [X9,oo+,X9],   [X9,ii+,X9],   [X9,zz+,X9]];
```

B Annex: Graph of the Universal Turing Program



C Annex: Universal Indirect Addressing Program

```
# INITIALISAT-
# ION OF THE
# REGISTERS
[0,cst,8,0],
[1,cst,10,2],
[2,cst,11,11],
[3,cst,12,20],
[4,cst,13,26],
[5,cst,14,33],
[6,cst,15,67],
[7,cst,16,68],
[8,cst,17,70],
[9,cst,18,76],
[10,cst,19,82],
[11,cst,20,88],
[12,cst,21,94],

# ENCODING OF
# THE EMULATED
# PROGRAM
# INITIALISAT-
# ION OF THE
# SOURCE POSIT-
# ION R[1] AND
# THE BOOLEAN
# VALUE c
[13,cst,2,24],
[14,cst,5,1],
[15,cst,0,16],
# INCREASING
# THE SOURCE
# POSITION R[1]
[16,plus,1,9],
# CASE STUDY
# ACCORDING
# TO THE VALUE
# a OF R[R[1]]
[17,from,3,1],
[18,to,1,8],
[19,plus,3,11],
[20,from,0,3],
# CASE a=1
[21,ifzero,5,24],
[22,cst,5,0],
[23,cst,4,0],
[24,plus,4,4],
[25,plus,4,9],
[26,cst,0,15],
# CASE a=2
[27,ifzero,5,30],
[28,cst,5,0],
[29,cst,4,0],
[30,plus,4,4],
[31,plus,4,9],
[32,plus,4,9],
[33,cst,0,15],
# CASE a=3
[34,ifzero,5,36],
[35,cst,0,40],
[36,plus,2,9],
[37,sub,4,9],
[38,to,2,4],
[39,cst,5,1],
[40,cst,0,15],
# END
[41,to,1,8],
[42,cst,4,1],
[43,plus,4,1],
[44,cst,6,25],
[45,to,4,6],

# PROGRAM
# EMULATION
# SKIP INCREM-
# ENTATION OF
# THE INSTRUC-
# TION COUNTER
[46,cst,0,49],
# INCREASING
# THE INSTRUC-
# TION COUNTER
[47,from,6,1],
[48,plus,6,9],
[49,to,1,6],
# COMPUTING THE
# POSITION OF
# INSTRUCTION
# NB ZERO
[50,from,7,1],
[51,cst,6,25],
[52,plus,6,7],
[53,plus,6,7],
[54,plus,6,7],
# HALTING TEST
[55,cst,7,0],
[56,plus,7,1],
[57,sub,7,6],
[58,ifzero,7,100],
# COMPUTING
# a:=R[R[6]]
[59,from,3,6],
# COMPUTING
# b:=R[R[6]]+R[1]
[60,plus,6,9],
[61,from,4,6],
[62,plus,4,1],
# COMPUTING
# c:=R[R[4]+2]
[63,plus,6,9],
[64,from,5,6],
# CASE STUDY
# ACCORDING TO
# THE VALUE OF a
[65,cst,6,15],
[66,plus,6,3],
[67,from,0,6],
# NO
# INSTRUCTION
[68,cst,0,99],
# CONSTANT
# INSTRUCTION
[69,to,4,5],
[70,cst,0,46],
# PLUS
# INSTRUCTION
[72,plus,5,1],
[73,from,5,5],
[74,plus,6,5],
[75,to,4,6],
[76,cst,0,46],
# MINUS
# INSTRUCTION
[77,from,6,4],
[78,plus,5,1],
[79,from,5,5],
[80,sub,6,5],
[81,to,4,6],
[82,cst,0,46],
# FROMINDIRECT
# INSTRUCTION
[83,plus,5,1],
[84,from,5,5],
[85,plus,5,1],
[86,from,5,5],
[87,to,4,5],
[88,cst,0,46],
# TOINDIRECT
# INSTRUCTION
[89,from,4,4],
[90,plus,4,1],
[91,plus,5,1],
[92,from,5,5],
[93,to,4,5],
[94,cst,0,46],
# IFZERO
# INSTRUCTION
[95,from,4,4],
[96,ifzero,4,98],
[97,cst,0,46],
[98,to,1,5],
[99,cst,0,49],
# END
[100,plus,9,1],
[101,from,9,9],
[102,plus,1,9].
```