# A language for Markov's algorithms composition

By G. GERMANO and A. MAGGIOLO—SCHETTINI

The present paper gives an improvement of [2]. There, after having noted that Markov's normal algorithms cannot be composed immediately like flow-charts, the authors presented four operations on Markov's normal algorithms without concluding formulas which helped to overcome the difficulty; among these operations there were the analogs of **if** and **goto**. Here other operations are given: the analogs of **if** and **goto** are substituted by the analog of **while**. This allows to use only one alphabet for output strings, whereas in [2] infinitely many alphabets are used (in the sense that two different algorithms might have as output alphabets $A^i$ and $A^j$ respectively with $i \neq j$), and to use a simpler definition of computability.

Furthermore an Algol-like language $L$ is given which is interpreted into Markov's normal algorithms without concluding formulas via the new operations defined for composing them. So the statements of $L$ come out to be names of Markov's normal algorithms and it is immediate to pass from traditional programming to algorithms. As a practical motivation, we have already mentioned the fact that this work gives the possibility of writing Markov's algorithms along the familiar patterns of computer programming. As a theoretical motivation, after having recalled the known relationship between programming and combinatory logic, we offer the following quotation from H.B. Curry [1]:

"Altough it is well known that any partial recursive numerical function can be represented in combinatory logic... and thus, by Church's thesis, any effective process can be so represented via the detour of Gödel representation, yet there is some interest in a direct representation, not involving this detour, of certain processes, like... Markov algorithms".

## § 1. Operations on algorithms

We will use alphabets $\{\star_i, |_i\}$ with $1 \leqq i \leqq \omega$ and will consider algorithms which transform words in the alphabet $\{\star_1, |_1\}$ into words in some alphabet $\bigcup_{i \in I} \{\star_i, |_i\}$ (where $I$ is finite for each algorithm) and eventually naturally transform such words into words in the alphabet $\{\star_\omega, |_\omega\}$ (see [3] for the notions of "transforms" and "naturally transforms"). We will use "$x_i$" and "$y_i$" to denote letters in the alphabet

$\{\curlywedge_i, |_i\}$. Analogously to [2], we will use the following translations for words in the alphabets above:

$$\curlywedge_i^{T_j} := \curlywedge_{i+j} \qquad |_i^{T_j} := |_{i+j}$$
$$\curlywedge_\omega^{T_j} := \curlywedge_\omega \qquad |_\omega^{T_j} := |_\omega$$
$$\curlywedge_i^{\tau_j} := \curlywedge_i \qquad |_i^{\tau_j} := |_i$$
$$\curlywedge_\omega^{\tau_j} := \curlywedge_j \qquad |_\omega^{\tau_j} := |_j$$

These translations are extended to words in the usual way and we will write $P_j$ to mean that each letter in $P$ is indexed by $j$. Algoritmhs are translated word by word.

We introduce three operations for composing algorithms, namely *juxtaposition, connection* and *controlled repetition* and prove the relative theorems.

1.1 **Juxtaposition.** The juxtaposition of the algorithms $\mathfrak{A}$ and $\mathfrak{B}$ is the algorithm

$$\begin{cases} x_1 \to x_2 x_{m+3} \\ y_{m+3} x_2 \to x_2 y_{m+3} \\ (\mathfrak{A}^{T_1})^{\tau_{m+2}} \\ (\mathfrak{B}^{T_{m+2}})^{\tau_{m+n+3}} \\ x_{m+2} \to x_\omega \\ x_{m+n+3} \to x_\omega \end{cases}$$

where $m=$maxind $\mathfrak{A}$ and $n=$maxind $\mathfrak{B}$ (maxind $\mathfrak{A}$ is defined as the highest index $i<\omega$ occurring in $\mathfrak{A}$, see [2]).

*Theorem.* If $\mathfrak{C}$ is the juxtaposition of $\mathfrak{A}$ and $\mathfrak{B}$ then for every $P\in\{\curlywedge_1, |_1\}^*$ it holds that

$$\mathfrak{C}(P) \cong \mathfrak{A}(P)\,\mathfrak{B}(P).$$

The proof is immediate (see [2] p. 305 for analogy).

1.2 **Controlled repetition.** The *repetition* of the algorithm $\mathfrak{B}$ *controlled* by the algorithm $\mathfrak{A}$ is the algorithm

$$\begin{cases} x_1 y_{m+n+3} \to x_1 y_1 & (1) \\ x_1 \to x_2 x_{m+3} & (2) \\ y_{m+3} x_2 \to x_2 y_{m+3} & (3) \\ (\mathfrak{A}^{T_1})^{\tau_{m+2}} & (4) \\ \curlywedge_{m+2} x_{m+2} y_{m+2} \to \curlywedge_{m+2} x_{m+2} & (5) \\ \curlywedge_{m+2} x_{m+2} y_{m+3} \to y_{m+3} & (6) \\ \curlywedge_{m+2} x_{m+3} \to x_\omega \curlywedge_{m+2} & (7) \\ \curlywedge_{m+2} \to & (8) \\ (\mathfrak{B}^{T_{m+2}})^{\tau_{m+n+3}} & (9) \\ \curlywedge_{m+n+3} \to \curlywedge_1 & (10) \end{cases}$$

where $m=$maxind $\mathfrak{A}$ and $n=$maxind $\mathfrak{B}$.

*Theorem.* If $\mathfrak{C}$ is the repetition of $\mathfrak{B}$ controlled by $\mathfrak{A}$ then for every $P \in \{\curlywedge_1, |_1\}^*$ it holds that

$$\mathfrak{C}(P) = \mathfrak{B}^j(P)$$

if $j$ is the least (non negative ) integer such that $\mathfrak{A}(\mathfrak{B}^j(P)) = \curlywedge_\omega$ whereas

$$\mathfrak{C}(P) \text{ is undefined}$$

if no such $j$ exists.

*Proof.* **I.** If for $i$ with $0 \leqq i < j$ $\mathfrak{A}(\mathfrak{B}^i(P)) \neq \curlywedge_\omega$ then it holds that

$$\begin{aligned}
&\mathfrak{C} : P \\
&\models\ P_2 P_{m+3} && \text{by (2), (3)} \\
&\models\ (\mathfrak{A}(P))_{m+2} P_{m+3} && \text{by (4)} \\
&\models\ P_{m+3} && \text{by (5), (6)} \\
&\models\ (\mathfrak{B}(P))_{m+n+3} && \text{by (9)} \\
&\models\ (\mathfrak{B}(P))_1 && \text{by (10), (1)} \\
&\quad \cdots \\
&\models\ (\mathfrak{B}^j(P))_1 .
\end{aligned}$$

**II.** If $\mathfrak{A}(P) = \curlywedge_\omega$ then it holds that

$$\begin{aligned}
&\mathfrak{C} : P \\
&\models\ P_2 P_{m+3} && \text{by (2), (3)} \\
&\models\ \curlywedge_{m+2} P_{m+3} && \text{by (4)} \\
&\models\ P_\omega \curlywedge_{m+2} && \text{by (7)} \\
&\models\ P_\omega \quad \neg && \text{by (8).}
\end{aligned}$$

From I and II the thesis of the theorem follows immediately.

1.3. **Connection.** The *connection* of the algorithms $\mathfrak{A}$ and $\mathfrak{B}$ is the algorithm

$$\begin{cases} \mathfrak{A}^{\tau_{m+1}} \\ \mathfrak{B}^{T_m} \end{cases}$$

where $m = \text{maxind } \mathfrak{A}$.

*Theorem.* If $\mathfrak{C}$ is the connection of $\mathfrak{A}$ and $\mathfrak{B}$ then for every $P \in \{\curlywedge_1, |_1\}^*$ it holds that

$$\mathfrak{C}(P) \cong \mathfrak{B}(\mathfrak{A}(P)).$$

The proof is immediate (see [2] p. 304 for analogy).

## § 2. The language

The syntactic definition of the language $L$ is

⟨initial statement⟩ := **zero**|**succ**|**pred**

⟨projection statement⟩ := **proj i**|⟨projection statement⟩**i**

⟨statement⟩ := ⟨initial statement⟩|⟨projection statement⟩|
(⟨statement⟩,⟨statement⟩)|
**while**⟨statement⟩**do**(⟨statement⟩|
(⟨statement⟩; ⟨statement⟩)

For short we will write $\mathbf{proj}^{(j)}$ instead of $\mathbf{proj}\ \overbrace{\mathbf{i} \ldots \mathbf{i}}^{j}$. As variables for statements we will use $\mathfrak{S}$ and $\mathfrak{T}$.

We define now an interpretation $\mathscr{I}$ of $L$ into Markov's normal algorithms without concluding formulas by induction on the syntactic definition of $L$:

$$\mathscr{I}(\mathbf{zero}) := \{ \ast_1 \to \ast_\omega$$

$$\mathscr{I}(\mathbf{succ}) := \begin{cases} \ast_1 \to \ast_\omega|_\omega \\ |_1 \to |_\omega \end{cases}$$

$$\mathscr{I}(\mathbf{pred}) := \begin{cases} \ast_1|_1 \to \ast_\omega \\ \ast_1 \to \ast_\omega \\ |_1 \to |_\omega \end{cases}$$

$$\mathscr{I}(\mathbf{proj}^{(j)}) := \begin{cases} \ast_\omega \ast_1|_1 \to \ast_\omega \ast_1 \\ |_\omega \ast_1|_1 \to |_\omega \ast_1 \\ \ast_\omega \ast_1 \to \ast_\omega \\ |_\omega \ast_1 \to |_\omega \\ \ast_\omega|_1 \to \ast_\omega|_\omega \\ |_\omega|_1 \to |_\omega|_\omega \\ \ast_2^j \to \ast_\omega \\ \ast_2 \ast_1 \to \ast_2 \ast_2 \\ \ast_2|_1 \to \ast_2 \\ \ast_1 \to \ast_2 \end{cases}$$

$\mathscr{I}((\mathfrak{S}, \mathfrak{T}))$ is defined as the juxtaposition of $\mathscr{I}(\mathfrak{S})$ and $\mathscr{I}(\mathfrak{T})$. $\mathscr{I}(\mathbf{while}\ \mathfrak{S}\ \mathbf{do}\ \mathfrak{T})$ is defined as the repetition of $\mathscr{I}(\mathfrak{T})$ controlled by $\mathscr{I}(\mathfrak{S})$. $\mathscr{I}((\mathfrak{S}; \mathfrak{T}))$ is defined as the connection of $\mathscr{I}(\mathfrak{S})$ and $\mathscr{I}(\mathfrak{T})$.

## § 3. Application

We say that a function $f$ is *computable* by $\mathfrak{S}$ (relatively to the input alphabet $\{\curlywedge_1, |_1\}$ and to the output alphabet $\{\curlywedge_\omega, |_\omega\}$) if and only if

$$\mathfrak{S}: \curlywedge_1 |_1^{n_1} \dots \curlywedge_1 |_1^{n_k} \vDash \curlywedge_\omega |_\omega^{f(n_1, \dots, n_k)} \neg.$$

As concerns computability of partial recursive functions (characterized as in [4]) it is immediate to see how to write programs for initial functions and concerning substitution. We give the programs concerning recursion scheme and $\mu$-operator.

Let the function $g$ be computable by $\mathfrak{S}$, the function $h$ be computable by $\mathfrak{T}$ and

$$\begin{cases} f(0, y) \cong g(y) \\ f(S(x), y) \cong h(y, f(x, y)) \end{cases}$$

Then the function $f$ is computable by

$$((((\mathbf{proj}^{(1)}, \mathbf{proj}^{(2)}), (\mathbf{proj}^{(2)}; \mathfrak{S}));$$
$$\mathbf{while}\ \mathbf{proj}^{(1)}\ \mathbf{do}\ (((\mathbf{proj}^{(1)}; \mathbf{pred}), \mathbf{proj}^{(2)}), ((\mathbf{proj}^{(2)}, \mathbf{proj}^{(3)}); \mathfrak{T})); \mathbf{proj}^{(3)}).$$

Let the function $g$ be computable by $\mathfrak{S}$ and

$$f(x_1, \dots, x_k) \cong \mu x(g(x, x_1, \dots, x_k) = 0)$$

Then the function $f$ is computable by

$$(((\dots((\mathbf{zero}, \mathbf{proj}^{(1)}), \dots, \mathbf{proj}^{(k)});$$
$$\mathbf{while}\ \mathfrak{S}\ \mathbf{do}\ (\dots((\mathbf{proj}^{(1)}; \mathbf{succ}), \mathbf{proj}^{(2)}), \dots, \mathbf{proj}^{(k+1)})); \mathbf{proj}^{(1)}).$$

So we may conclude that every partial recursive function is computable by Markov's normal algorithms without concluding formulas relatively to the input alphabet $\{\curlywedge_1, |_1\}$ and to the output alphabet $\{\curlywedge_\omega, |_\omega\}$.


## Abstract

A programming language is introduced to whose statements Markov's algorithms univocally correspond via the operations of juxtaposition, connection and controlled repetition. Avoiding goto statements allows to use only one output alphabet.



LABORATORIO DI CIBERNETICA
DEL C. N. R.
80072 ARCO FELICE, ITALY

INSTITUTO DI SCIENZE DELL'INFORMAZIONE
DELL'UNIVERSITA, VIA VERNIERI 42
84100 SALERNO, ITALY


## References


[1] CURRY, H. B., Representation of Markov algorithms by combinators, *Notices Amer. Math. Soc.*, v. 20, A-590, 1973.

[2] GERMANO, G. & A. MAGGIOLO-SCHETTINI, A flow diagram composition of Markov's normal algorithms without concluding formulas, *BIT*, v. 13, 1973, pp. 301—312.

[3] MARKOV, A. A., Teoria algoritmov (Russian), *Trudy Math. Inst. Steklov.*, v. 42, 1954.

[4] ROBINSON, R. M., Primitive recursive functions, *Bull. Amer. Math. Soc.*, v. 53, 1947, pp. 925—942.


*(Received Oct. 7, 1974)*