APcol systems and Turtle Graphics

Lucie Ciencialová, Luděk Cienciala

Institute of Computer Science, Silesian University in Opava, Czech Republic
lucie.ciencialova@fpf.slu.cz
ludek.cienciala@fpf.slu.cz

Abstract: In this paper we continue our investigations in APCol systems (Automaton-like P colonies), variants of P colonies where the environment of the agents is given by a string and the functioning of the system resembles to the functioning of standard finite automaton. We introduce a new variant of APcol systems called extended APcol systems - agents capacity is not limited to two objects. We deal with the concept of agent creation in these systems and possibility of generation of strings as commands for turtle graphics.

1 Introduction

Automaton-like P colonies (APCol systems, for short), introduced in [2], are variants of P colonies (introduced in [9]) - very simple membrane systems inspired by colonies of formal grammars. The interested reader is referred to [11] for detailed information on membrane systems (P systems) and to [10] and [6] for more information to grammar systems theory. For more details on P colonies consult the surveys [8] and [5].

An APCol system consists of a finite number of agents and their joint shared environment. The agents are formed from a finite number of objects and their functioning is based on programs consisting of rules. These rules are of two types: they may change the objects of the agents and they can be used for interacting with the joint shared environment of the agents. While in the case of standard P colonies the environment is a multiset of objects, in case of APCol systems it is represented by a string. The number of objects inside each agent is set by definition and it is usually a very small number: 1, 2 or 3. The string representing the environment is processed by the agents and it is used as a communication channel for the agents as well, since through the string, the agents are able to affect the behaviour of another agent.

Agents are also equipped with agent creation programs. They are applicable when a special object appears inside the agent. An agent with the special object can make one copy of itself containing objects specified by the program.

The computation in APCol systems starts with an input string, representing the environment, and with each agent in its initial state.

Every computational step is done in a maximally parallel way which means that a set of the active agents performing their programs is maximal - the joint action of the agents is maximally parallel if no more active agent can be added to the synchronously acting agents. An agent is active if it is able to execute at least one of its programs.

The computation ends if there are no more applicable programs in the system.

For more detailed information on APCol systems we refer to [3, 4].

In computer graphics, turtle graphics are vector graphics using a relative cursor - the “turtle” to draw curves. Commands for turtle can be formed into string and hence we can speak about a turtle representation of string. A state of the turtle is defined as a triplet (x,y,α), where the Cartesian coordinates (x,y) represent the turtle's position, and the angle α, called the heading, is interpreted as the direction in which the turtle is facing. Given the step size d and the angle increment δ, the turtle can respond to commands represented by the following symbols:

F Move forward a step of length d. The state of the turtle changes to (x',y',α), where $x' = x + d\cos\alpha$ and $y' = y + d\sin\alpha$. A line segment between points (x,y) and (x',y') is drawn.

f Move forward a step of length d without drawing a line.

$+$ Turn left by angle δ. The next state of the turtle is $(x,y,\alpha+\delta)$. The positive orientation of angles is counter-clockwise.

$-$ Turn right by angle δ. The next state of the turtle is $(x,y,\alpha-\delta)$.

Given a string s, the initial state of the turtle (x_0,y_0,α_0) and fixed parameters d and δ, the turtle interpretation of s is the curve (set of lines) drawn by the turtle in response to the string s. Turtle graphics is used for example to interpret strings generated by L-systems (more details reader can find in [13]).

In this paper we deal with APcol systems with agent creation programs generating strings that can be interpreted in turtle graphics. We construct APcol system generating string representation of two space-filling curves Hilbert curve and Peano curve. In [1] the authors deal with generation of string representation of space-filling curves by P systems with array rewriting rules.

2 Preliminaries and Basic Notions

Throughout the paper we assume the reader to be familiar with the basics of the formal language theory and membrane computing [14, 11].

For an alphabet Σ, the set of all words over Σ (including the empty word, ε), is denoted by Σ^*. We denote the length of a word $w \in \Sigma^*$ by $|w|$ and the number of occurrences of the symbol $a \in \Sigma$ in w by $|w|_a$.

A multiset of objects M is a pair $M = (O, f)$, where O is an arbitrary (not necessarily finite) set of objects and f is a mapping $f : O \to N$; f assigns to each object in O its multiplicity in M. Any multiset of objects M with the set of objects $O = \{x_1, \ldots x_n\}$ can be represented as a string w over alphabet O with $|w|_{x_i} = f(x_i)$; $1 \leq i \leq n$. Obviously, all words obtained from w by permuting the letters can also represent the same multiset M, and ε represents the empty multiset.

2.1 Extended APCol System

APCol system is a kind of P colony. It is formed from agents with capacity 2 processing the string. They act according to programs as it is usual in P colonies. Each program is formed from two rules of two types. The first type called rewriting is of the form $a \to b$ and by execution of this rule, the object a inside the agent is rewritten to the object b. The second type of rules is called communication. The communication rules are of the form $c \leftrightarrow d$ and by use of them, the agent replaces the symbol d in the string by object c initially placed inside the agent. If $c = e$ ($e \leftrightarrow d$) it means that agent erases symbol d from the string. If $d = e$ ($c \leftrightarrow e$) it means that agent inserts symbol c to the string in a place which is determined by use of another communication rule in the same program or if there is no other communication rule in the program c is placed in random position.

Let us make a few comments about the application of the programs.

1. Agent can act in only one place in the string in one step of computation.
2. $\langle a \leftrightarrow b; c \leftrightarrow e \rangle \quad \Rightarrow \quad b \to ac$ in the string,

   $\langle c \leftrightarrow e; a \leftrightarrow b \rangle \quad \Rightarrow \quad b \to ca$ in the string,

   $\langle a \to b; c \leftrightarrow e \rangle \quad \Rightarrow \quad \varepsilon \to c$ in the string, insert c anywhere to the string, and rewrite a to b inside agent

   $\langle a \to b; e \leftrightarrow d \rangle \quad \Rightarrow \quad d \to \varepsilon$ in the string, delete one d from the string, and rewrite a to b inside agent

If an agent contains a special object @, the agent makes a copy of itself. This action is done by executing a program formed from two rewriting rules. Let $a@$ be a contents of agent A_1 with program $p_1 = \langle @ \to b; a \to c \rangle$. After execution of the program p_1 there is one new child-agent in the APCol system with the same label and the same set of programs as the parent-agent A_1 has. The contents of the parent-agent after the execution of the program is bc while the contents of the child-agent is ba.

If the parent-agent has a program $p_2 = \langle a \to c; @ \to b \rangle$, then after the execution of the program p_2 the contents of parent-agent after the execution of the program is bc and the contents of the child-agent is bc, too.

The order of rules determines whether the rewriting rule without @ is used before or after the creation of the child-agent.

An extended APcol system is APcol system having capacity $k \geq 1$ given by definition. The increase of the capacity results in an elongation of the context for the application of programs with communication rules. In programs with agent creation rules, the increased capacity affects the content of agents - the rules listed before the agent creation rule are executed before the creation of a new agent, and the rules listed after the agent creation rule are executed only on the parent agent.

We introduce new type of rule, object occurrence conditional rule. The rule (rewritting, communication) is enriched by the prefix "$o|$" and it means that the rule can be executed only if object o is present in the environmental string in current configuration under the assumption that the rule itself is applicable. Prefix "$e|$" means that rule is applicable if rule without prefix is applicable too (it adds no applicability condition). We also define prefix "$\neg o|$" and it means that the rule can be executed only if object o is not present in the environmental string in current configuration under the assumption that the rule itself is applicable.

Definition 1. *An extended APCol system (eAPcol system for short) with capacity $k \geq 1$ with agent creation is a construct*

$$\Pi = (O, e, @, A_1, \ldots, A_n),$$

where

- *O is an alphabet, its elements are called the objects;*
- *$e \in O$, called the basic object;*
- *$@ \in O$, called the agent creation object;*
- *A_i, $1 \leq i \leq n$, are agents. Each agent is a triplet $A_i = (\omega_i, P_i, F_i)$, where*
  - *ω_i is a multiset over O, describing the initial state (contents) of the agent, $|\omega_i| = k$,*
  - *$P_i = \{p_{i,1}, \ldots, p_{i,k_i}\}$ is a finite set of programs associated with the agent, where each program is a pair of rules. Each rule is in one of the following forms:*
    - *$a \to b$, where $a, b \in O$, called an rewriting rule,*
    - *$c \leftrightarrow d$, where $c, d \in O$, called a communication rule;*

If @ appears on the left side of the rule in the program, all rules of this program must be rewriting; Prefix "o|" can be add to rule (not with @ on the left side) and it adds condition for applicability of the program - object $o \in O$ must be present in the environmental string. Prefix "¬o|" can be add to rule (not with @ on the left side) and it means that program is applicable if object $o \in O$ is not present in the environmental string.

– *$F_i \subseteq O^*$ is a finite set of final states (contents) of agent A_i.*

When an agent obtains the object @ by the execution of a rewriting or a communication rule in the program, the agent must create a new agent in the next step of the computation in the way described within the above definition.

The computation starts in the initial configuration where all agents contain their initial multiset of objects and there is an input string over the alphabet T on the eAPCol system. Consequently, an initial configuration of the eAPCol system is an $(n+1)$-tuple $c = (\omega; \omega_1, \ldots, \omega_n)$ where ω is the initial state of the environment and the other n components are multisets of objects, given in the form of strings, the initial states of the agents.

A configuration of an eAPCol system Π is given by $(w; w_1, \ldots, w_{n_l})$, where $|w_i| = k$, $1 \leq i \leq n$, w_i represents all the objects inside the agent labelled A_i and $w \in (O - \{e\})^*$ is the string to be processed.

At each step of the computation every agent attempts to find one of its programs to use. If the number of applicable programs is higher than one, then the agent nondeterministically chooses one of them. At one step of computation, the maximal possible number of agents have to be active, i.e., have to perform a program.

By executing programs, the eAPCol system passes from one configuration to another configuration. A sequence of configurations started from the initial configuration is called a computation.

The computation halts when no agent has an applicable program.

3 Space-Filling Curves generation by eAPcol Systems

In this paper, we focus on the generation of control strings for generating space-filling curves using turtle graphics. We concentrated on three curves - Hilbert, Moore and Peano. All three curves are fractal-like. A new iteration of the curve is formed according to a given rule from the previous iteration. This increases the area filled by the curve.

Various mechanisms are used to generate iterations of these curves (or control strings for turtle graphics), for example, parallel grammars or L-systems.

There are several techniques for generating these strings by eAPcol systems. We can use a sequential approach and follow the rules of parallel grammars to rewrite all occurrences of non-terminals one by one, so that the agent " moves" the object-tag that separates the processed part of the string from the unprocessed part as it rewrites non-terminals to strings.

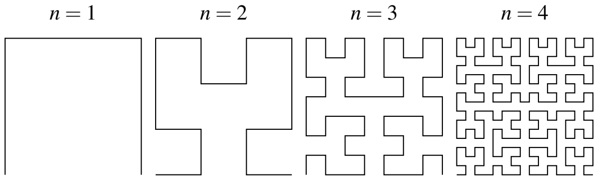

Figure 1: First four approximations of the Hilbert curve, H_1, H_2, H_3, H_4.

Alternatively, we can use a parallel approach (as in L-systems [13]) that requires rewriting all non-terminals in string in one step. Because of the increasing number of non-terminals, the number of agents that process the string needs to increase.

In this paper, we use both approaches. We use the parallel approach to generate the string describing the curve and the transcoding of the result into the turtle graphics language is done sequentially.

3.1 Hilbert curve

In 1891 Hilbert introduced the Hilbert space-filling curve in [7]. Its finite approximations, the Hilbert words, were first described as chain-code words in [15]. The first approximation is given by $H_1 = urd$ where u, d, r, l are objects representing direction up, down, right and left, and, for $n > 1$; the subsequent approximations are given by

$$H_{n+1} = \gamma(H_n) u H_n r H_n d \varphi(H_n)$$

where γ, φ are homomorphisms on alphabet given by:
$\gamma(u) = r; \gamma(d) = l; \gamma(l) = d; \gamma(r) = u$ – symmetry w.r.t. O_y and rotation right 90 degrees;
$\varphi(u) = l; \varphi(d) = r; \varphi(l) = u; \varphi(r) = d$ – symmetry w.r.t. O_y and rotation left 90 degrees.

The first four approximations are shown in Figure 1.

Hilbert words can be generated by context-free rules that are applied parallelly to string formed from terminals and non-terminals. To generate Hilbert words we use rewriting context-free (CF) rules over non-terminal alphabet $\{N, L, R, U\}$ and terminals from $\{u, d, l, r\}$. The CF rules are:

1. $N \rightarrow LuNrNdR$

2. $L \rightarrow NrLuLlU$

3. $R \rightarrow UlRdRrN$

4. $U \rightarrow RdUlUuL$

The starting string is N. In the last step of generation of Hilbert word we use another set of rules that is formed from ε-rules and all non-terminals are erased from word. Another approach is to count with terminals only in each generated word. Hilbert word H_1 is obtained by execution of the first rule to starting string

$$N \Rightarrow LuNrNdR \Rightarrow urd$$

The second Hilbert word, H_2 is generated from N by following derivation:

$$\begin{array}{ll} N \Rightarrow & LuNrNdR \Rightarrow \\ \Rightarrow & NrLuLlUuLuNrNdRrLuNrNdRdUlRdRrN \Rightarrow \\ \Rightarrow & ruluurdrurddldr \end{array}$$

For the turtle to draw the first approximation of the Hilbert curve, the command string is $F-F-F$. The command string for second approximation of Hilbert curve is $F+F+F-FF-F-F+F+F-F-FF-F+F+F$.

To generate Hilbert curve we construct eAPcol system with capacity three and with agent creation formed from two kind of agents.

1. The input of the system is a string $s_0 = a^n\#$ corresponding to the number of iterations concatenated with a special symbol labelling the end of input. The output of system is command string for Turtle Graphics to generate n-th approximation of Hilbert curve concatenated with symbol $\#_2$.

2. The first agent A_1 removes one occurrence of object a after all non-terminals are deleted from a string of computation.

3. The second agent A_2 will generates approximations of Hilbert word in two phases.

   In the first phase the agent generates symbols corresponding to the direction of movement of turtle (u, d, r, l for direction up, down, right and left) with objects called non-terminals according to CF rules.

   In the second phase, the agent makes a copy of itself twice. It means that in three steps the number of agents labelled by A_2 rises four times (the reason is the parallel simulation of the execution of the rules listed below).

4. The last phase of computation is to "code" directions u, d, l, r into turtle language.

Now, we describe the function of the constructed system in more detail, present the programs of the agents, and illustrate the function of the system using the example of generating a command string for the first approximation of the Hilbert curve.

Agent A_1 is equipped by the following programs for first three phases of computation:

$$\begin{array}{ll}
A1 & < \# | e \to e; e \to e; e \to b >; \\
A2 & < e \to e; e \to e; b \to c >; \\
A3 & < \neg N | e \to w_1; e \to e; c \to e >; \\
A4 & < \neg L | w_1 \to w_2; e \to e; e \to e >; \\
A5 & < \neg R | w_2 \to w_3; e \to e; e \to e >; \\
A6 & < \neg U | w_3 \to w_4; e \to e; e \to e >; \\
A7 & < a | w_4 \to 0; e \to g; e \to e >; \\
A8 & < 0 \to 1; g \leftrightarrow a; e \to e >; \\
A9 & < \neg a | w_4 \to 0'; e \to f; e \to e >; \\
A10 & < 0' \to 0'; f \leftrightarrow \#_1; e \to e >; \\
A11 & < a \to e; \textcircled{1} \to \textcircled{2}; e \leftrightarrow g >; \\
A12 & < g \to e; \textcircled{2} \to \textcircled{3}; e \to e >; \\
A13 & < e \to e; \textcircled{i} \to \textcircled{i+1}; e \to e >; 3 \leq i \leq 14 \\
A14 & < \textcircled{15} \to e; e \to e; e \to c >
\end{array}$$

The second agent in the first two steps generates the starting word N. The agent places object N next to object #. The communication rule is equipped with condition a :, it means that agent can put N into string only if a is present in the sting. This is initialization phase for generation of Hilbert word.

$$\begin{array}{ll}
B1 & < e \to \#_1, e \to N; e \to e >; \\
B2 & < \#_1 \leftrightarrow \#; a | N \leftrightarrow e; e \to e >
\end{array}$$

In our example, the initial configuration of the system is $(a\#; eee_{A_1}, eee_{A_2})$. The first phase of computation is shown in the Table 1. In rows, there are configurations of the system and sets of applicable programs. If there are more than one applicable program associated with an agent, the executed program is indicated by bold.

Configuration			Applicable programs	
Env.	A_1	A_2	A_1	A_2
$a\#$	eee	eee	$A1$	$B1$
$a\#$	eeb	$\#_1Ne$	$A2$	$B2$
$a\#_1N$	eec	$\#ee$	$-$	$B3$

Table 1: The first phase of generation of command string for H_1

In the next phase, the agent A_2 replaces N by string $LuNrNdR$ in accordance to the first CF rule. First it generates object $\#_X$, where X is non-terminal object. This program in non-deterministically chosen. If there is no such non-terminal present in the string (before this step there was no such non-terminal or the number of occurrences of non-terminal X was less than the number of agents that generated object $\#_X$), the agent can rewrite object $\#_X$ into # in the next step. It is done in fourteen steps, because agent uses program with two rewriting rules, then another one with two communication rules to generate one object of the right side of rule and one object marking the point where to insert new objects. We note that all CF rules are of the same length and all the agents are working simultaneously so when there are more agents with label

A_2 they can insert object in different parts of string. But the agents can consume object marker (for example $\#_{N_2}$ - marker object for insertion of second object of CF rule with N on the left-hand side of the rule) that corresponds only to currently performed CF rule. Let $X \in \{N, L, R, U\}$ and $X \to x_1x_2x_3x_4x_5x_6x_7$ is CF rule.

$B3$ $\quad < \# \to \#_X, e \to e; e \to e >;$

$B4$ $\quad < \#_X \leftrightarrow X; e \to \#_{X_1}; e \to e >$

$B5$ $\quad < \neg X | \#_X \to \#, e \to e; e \to e >;$ For a case that there is no non-terminal X in the string

$B6$ $\quad < \#_{X_1} \to \#_{X_1}, X \to x_1; e \to X_1 >;$

$B7$ $\quad < g | x_1 \leftrightarrow \#_X; \#_{X_1} \leftrightarrow e; X_1 \to X_1 >;$

$B8$ $\quad < \#_X \to \#_{X_2}, e \to x_2; X_1 \to X_2 >;$

$B9$ $\quad < x_2 \leftrightarrow \#_{X_1}; \#_{X_2} \leftrightarrow e; X_2 \to X_2 >$

$B10$ $\quad < \#_{X_{i-2}} \to \#_{X_i}, e \to x_i; X_{i-1} \to X_i >; \quad 3 \leq i \leq 7$

$B11$ $\quad < x_i \leftrightarrow \#_{X_{i-1}}; \#_{X_i} \leftrightarrow e; X_i \to X_i >; \quad 3 \leq i \leq 6$

$B12$ $\quad < x_7 \leftrightarrow \#_{X_6}; \#_{X_7} \to e; X_7 \to X_7 >;$

$B13$ $\quad < \#_{X_6} \to @, e \to F; X_7 \to e >$

We demonstrate the application of the programs by continuing the example: In Table 2, we show how the computation is done when agent A_2 chooses program associated with non-terminal that is not present in the string (or non-terminal is used by another copy of the agent).

Configuration			Applicable programs	
Env.	A_1	A_2	A_1	A_2
$a\#_1N$	eec	$\#ee$	$-$	$B3 : <\# \to \#_L, e \to e; e \to e>$
$a\#_1N$	eec	$\#_Lee$	$-$	$B5 : <\neg L | \#_L \to \#, e \to e; e \to e>$
$a\#_1N$	eec	$\#ee$	$-$	$B3$

Table 2: The second phase of generation of command string for H_1 - wrong choice of non-terminal

In Table 3 the computation with right choice of program is shown.

After insertion of all the objects from the right-hand side of the CF rule, the agent A_2 is creating new agents in two steps using programs

$B14$ $\quad < F \to S; e \to @; @ \to e; >;$

$B15$ $\quad < S \to e; e \to e; @ \to \# >$

After this phase in the first iteration the environmental string is in the form $a^{n-1}\#_1LuNrNdR$ and there are four agents labelled A_2 in the eAPcol system.

In Table 5, the creation of new agents labelled A_2 is shown.

When agent A_1 deletes all occurrences of object a from the environmental string, f appears in the environment and agents labelled A_2 delete all non-terminals from the string.

$B16$ $\quad < f | x_1 \to f; \#_{X_1} \to e; X_1 \to X_1 >;$

$B17$ $\quad < f \to f; e \leftrightarrow \#_X; X_1 \to f >;$

If the environmental string was $\#_1LuNrNdR$ before this phase of computation, four agents deleted non-terminals from the string and after this phase the string is $furd$.

In final phase the agent A_1 rewrite objects $\{u, d, l, r\}$ to symbols $\{F, +, -\}$. The coding depends on two consecutive objects, for example if ru is substring of the environmental string u is replaced by $+F$ because to draw line in direction up after line in direction right turtle must turn 90 degrees left and draw a line.

To draw Hilbert curve by turtle we have to set two parameters - the step size and the angle increment - and initial state of the turtle. For code generation it is necessary to set the angle increment $\delta = 90°$ and turtle orientation to up ($\alpha = 90°$). Because of angle increment the turtle can be oriented in four directions (up, down, right, left). Boxed symbols $\left(\boxed{u}, \boxed{d}, \boxed{r}, \boxed{l} \right)$ can represent current turtle orientation.

There are following programs in a set of programs of agent A_1:

$A15$ $\quad < 0' \to \boxed{u}; \#_1 \to \#_2; e \to e >;$

to move in the same direction as turtle is facing

$x \in \{u, d, r, l\}$

$A16$ $\quad < \boxed{x} \to \boxed{x}; \#_2 \leftrightarrow f; e \leftrightarrow x >;$

$A17$ $\quad < \boxed{x} \to \boxed{x}; f \to f; x \to F >;$

$A18$ $\quad < \boxed{x} \to \boxed{x}; F \leftrightarrow \#_2; f \leftrightarrow e >;$

to rotate and move

$cc(u) = r; cc(r) = d; cc(d) = l; cc(l) = u; p = cc(x)$

$A19$ $\quad < \boxed{x} \to \boxed{x}; \#_2 \leftrightarrow f; e \leftrightarrow p >;$

$A20$ $\quad < \boxed{x} \to \boxed{p}; f \to f'; p \to - >;$

$A21$ $\quad < \boxed{p} \to \boxed{p}; - \leftrightarrow \#_2; f' \leftrightarrow e >;$

$A22$ $\quad < \boxed{p} \to \boxed{p}; \#_2 \leftrightarrow f'; e \to F >;$

$A23$ $\quad < \boxed{p} \to \boxed{p}; f' \to f; F \to F >;$

$cw(u) = l; cw(l) = d; cw(d) = r; cw(r) = u; q = cw(x)$

$A24$ $\quad < \boxed{x} \to \boxed{x}; \#_2 \leftrightarrow f; e \leftrightarrow q >;$

$A25$ $\quad < \boxed{x} \to \boxed{q}; f \to f'; q \to + >;$

$A26$ $\quad < \boxed{q} \to \boxed{q}; + \leftrightarrow \#_2; f' \leftrightarrow e >;$

$A27$ $\quad < \boxed{q} \to \boxed{q}; \#_2 \leftrightarrow f'; e \to F >;$

$A28$ $\quad < \boxed{q} \to \boxed{q}; f' \to f; F \to F >;$

to rotate twice and move

$dc(u) = d; dc(l) = r; dc(d) = u; dc(r) = l; y = dc(x)$

$A29$ $\quad < \boxed{x} \to \boxed{x}; \#_2 \leftrightarrow f; e \leftrightarrow y >;$

$A30$ $\quad < \boxed{x} \to \boxed{y}; f \to f''; y \to + >;$

$A31$ $\quad < \boxed{y} \to \boxed{y}; + \leftrightarrow \#_2; f'' \leftrightarrow e >;$

$A32$ $\quad < \boxed{y} \to \boxed{y}; \#_2 \leftrightarrow f''; e \to + >;$

$A33$ $\quad < \boxed{y} \to \boxed{y}; f'' \to f'; + \to + >;$

If the initial angle of the turtle is 90 degrees (it faces up) the agent A_1 rewrites string into $F - F - F$. If the initial angle of the turtle is 0 degrees (it faces right) the agent A_1 rewrites string into $+F - F - F$. The initial angle is generated by a program that launches this phase of computation and it is given by definition of used turtle graphics system.

Configuration			Applicable programs	
Env.	A_1	A_2	A_1	A_2
$a\#_1N$	eec	$\#ee$	–	$B3(X{=}N)$
$a\#_1N$	eec	$\#_Nee$	–	$B4$
$a\#_1\#_N$	eec	$N\#_{N_1}ee$	$A3$	$B6$
$a\#_1\#_N$	w_1ee	$\#_{N_1}LN_1$	$A4$	–
$a\#_1\#_N$	w_2ee	$\#_{N_1}LN_1$	$A5$	–
$a\#_1\#_N$	w_3ee	$\#_{N_1}LN_1$	$A6$	–
$a\#_1\#_N$	w_4ee	$\#_{N_1}LN_1$	$A7$	–
$a\#_1\#_N$	$0ge$	$\#_{N_1}LN_1$	$A8$	–
$g\#_1\#_N$	$\textcircled{1}ae$	$\#_{N_1}LN_1$	$A11$	$B7(x_1{=}L)$
$\#_1L\#_{N_1}$	$\textcircled{2}eg$	$\#_NeN_1$	$A12$	$B8(x_2{=}u)$
$\#_1L\#_{N_1}$	$e\textcircled{3}e$	$\#_{N_2}uN_2$	$A13(i{=}3)$	$B9$
$\#_1Lu\#_{N_2}$	$e\textcircled{4}e$	$\#_{N_1}eN_2$	$A13(i{=}4)$	$B10(i{=}3;\ x_3{=}N)$
$\#_1Lu\#_{N_2}$	$e\textcircled{5}e$	$\#_{N_3}NN_3$	$A13(i{=}5)$	$B11(i{=}3)$
$\#_1LuN\#_{N_3}$	$e\textcircled{6}e$	$\#_{N_2}eN_3$	$A13(i{=}6)$	$B10(i{=}4;\ x_4{=}r)$
$\#_1LuN\#_{N_3}$	$e\textcircled{7}e$	$\#_{N_4}rN_4$	$A13(i{=}7)$	$B11(i{=}4)$
$\#_1LuNr\#_{N_4}$	$e\textcircled{8}e$	$\#_{N_3}eN_4$	$A13(i{=}8)$	$B10(i{=}5;\ x_5{=}N)$
$\#_1LuNr\#_{N_4}$	$e\textcircled{9}e$	$\#_{N_5}NN_5$	$A13(i{=}9)$	$B11(i{=}5)$
$\#_1LuNrN\#_{N_5}$	$e\textcircled{10}e$	$\#_{N_4}eN_5$	$A13(i{=}10)$	$B10(i{=}6;\ x_6{=}d)$
$\#_1LuNrN\#_{N_5}$	$e\textcircled{11}e$	$\#_{N_6}dN_6$	$A13(i{=}11)$	$B11(i{=}6)$
$\#_1LuNrNd\#_{N_6}$	$e\textcircled{12}e$	$\#_{N_5}eN_6$	$A13(i{=}12)$	$B10(i{=}7;\ x_7{=}R)$
$\#_1LuNrNd\#_{N_6}$	$e\textcircled{13}e$	$\#_{N_7}RN_7$	$A13(i{=}13)$	$B12$
$\#_1LuNrNdR$	$e\textcircled{14}e$	$\#_{N_6}eN_7$	$A13(i{=}14)$	$B13$
$\#_1LuNrNdR$	$e\textcircled{15}e$	$@Fe$	$A14$	$B14$

Table 3: The second phase of generation of command string for H_1 - right choice of non-terminal

Configuration						Applicable programs				
Env.	A_1	A_2	A_2	A_2	A_2	A_1	A_2	A_2	A_2	A_2
$\#_1LuNrNdR$	$e\textcircled{15}e$	$@Fe$	–	–	–	$A14$	$B14$			
$\#_1LuNrNdR$	eec	$S@e$	$S@e$	–	–	–	$B15$	$B15$		
$\#_1LuNrNdR$	eec	$ee\#$	$ee\#$	$ee\#$	$ee\#$	–	$B3$	$B3$	$B3$	$B3$

Table 4: The third phase of generation of command string for H_1 - agent creation

3.2 Moore curve

The Moore curve is a variant of Hilbert curve. The first four iterations are depicted in Figure 2. To generate word representation of Moore curve we use rewriting context-free (CF) rules over non-terminal alphabet $\{A,B,C,E,F\}$ and terminals from $\{u,d,l,r\}$. The CF rules are:

1. $A \rightarrow BuBrGdG$
2. $B \rightarrow ClBuBrE$
3. $C \rightarrow BuClCdG$
4. $E \rightarrow GdErEuB$
5. $G \rightarrow ErGdGlC$

A derivation of word representation of Moore curve starts with string A. The derivation of word M_1 is

$$A \Rightarrow BuBrFdF \Rightarrow urd$$

and the word M_2 is generated as follows:

$$\begin{aligned} A \Rightarrow \quad & BuBrGdG \Rightarrow \\ \Rightarrow \quad & ClBuBrEuClBuBrErErGdGlCdErGdGlC \Rightarrow \\ \Rightarrow \quad & lurulurrrdldrdl \end{aligned}$$

eAPcol systems generating Moore curve are very similar to that one generating commands for Hilbert curve. They differs only in the programs for the second phase of part 3, because the generating CF rules are different. But the CF rules are of the same length and they contain the same number of non-terminals on their right hand side.

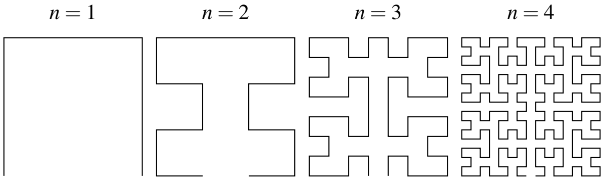

Figure 2: First four approximations of the Moore curve, M_1, M_2, M_3, M_4.

3.3 Peano Curve

Peano defined this space-filling curve in 1890, in [12].

Represented as chain-code words, the first Peano word is $P_1 = uurddruu$, and for $n > 1$; the subsequent approximations are given by

$$P_{n+1} = P_nu\lambda(P_n)uP_nr\rho(P_n)d\delta(P_n)d\rho(P_n)rP_nu\lambda(P_n)uP_n$$

where $\lambda;\rho;\delta$ are homomorphisms on the alphabet of directions$\{l;r;u;d\}$ as given below:
$\lambda(u) = u;\lambda(d) = d;\lambda(l) = r;\lambda(r) = l$ – symmetry w.r.t. O_y axis
$\rho(u) = d;\rho(d) = u;\rho(l) = l;\rho(r) = r$ – symmetry w.r.t. O_x axis
$\delta(u) = d;\delta(d) = u;\delta(l) = r;\delta(r) = l$ – symmetry w.r.t. origin.

The constructed eAPcol system is very similar to the one that generates representation of Hilbert curve. First let us introduce CF rules generating Peano curve direction representation. The CF rules are:

Configuration						Applicable programs				
Env.	A_1	A_2	A_2	A_2	A_2	A_1	A_2	A_2	A_2	A_2
$\#_1 LuNrNdR$	eec	$ee\#$	$ee\#$	$ee\#$	$ee\#$	–	$B3(X=L)$	$B3(X=N)$	$B3(X=N)$	$B3(X=R)$
$\#_1 LuNrNdR$	eec	$ee\#_L$	$ee\#_N$	$ee\#_N$	$ee\#_R$	–	$B4$	$B4$	$B4$	$B4$
$\#_1\#_L u\#_N r\#_N d\#_R$	eec	$L\#_{L_1}e$	$N\#_{N_1}e$	$N\#_{N_1}e$	$R\#_{R_1}e$	$A3$	$B6$	$B6$	$B6$	$B6$
$\#_1\#_L u\#_N r\#_N d\#_R$	eew_1	$N\#_{L_1}L_1$	$L\#_{N_1}N_1$	$L\#_{N_1}N_1$	$U\#_{R_1}R_1$	$A4$	–	–	–	–
$\#_1\#_L u\#_N r\#_N d\#_R$	eew_2	$N\#_{L_1}L_1$	$L\#_{N_1}N_1$	$L\#_{N_1}N_1$	$U\#_{R_1}R_1$	$A5$	–	–	–	–
$\#_1\#_L u\#_N r\#_N d\#_R$	eew_3	$N\#_{L_1}L_1$	$L\#_{N_1}N_1$	$L\#_{N_1}N_1$	$U\#_{R_1}R_1$	$A6$	–	–	–	–
$\#_1\#_L u\#_N r\#_N d\#_R$	eew_4	$N\#_{L_1}L_1$	$L\#_{N_1}N_1$	$L\#_{N_1}N_1$	$U\#_{R_1}R_1$	$A9$	–	–	–	–
$\#_1\#_L u\#_N r\#_N d\#_R$	$ef0'$	$N\#_{L_1}L_1$	$L\#_{N_1}N_1$	$L\#_{N_1}N_1$	$U\#_{R_1}R_1$	$A10$	–	–	–	–
$f\#_L u\#_N r\#_N d\#_R$	$e\#_1 0'$	$N\#_{L_1}L_1$	$L\#_{N_1}N_1$	$L\#_{N_1}N_1$	$U\#_{R_1}R_1$	$A15$	$B16$	$B16$	$B16$	$B16$
$f\#_L u\#_N r\#_N d\#_R$	$\boxed{u}\#_2 e$	feL_1	feN_1	feN_1	feR_1	–	$B17$	$B17$	$B17$	$B17$
$furd$	$\boxed{u}\#_2 e$	$ff\#_L$	$ff\#_N$	$ff\#_N$	$ff\#_R$	$A16$	–	–	–	–

Table 5: The fourth phase of generation of command string for H_1 - deletion of all non-terminals

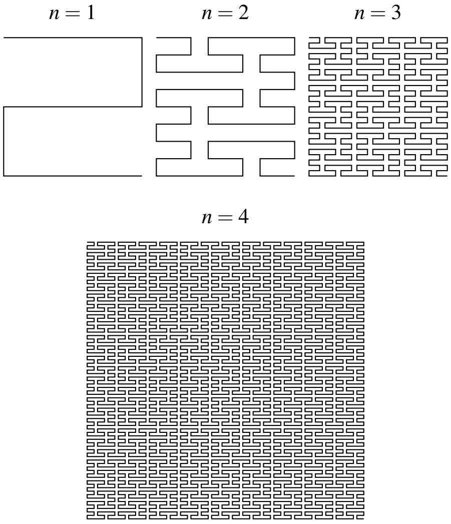

Figure 3: First four approximations of the Peano curve, P_1, P_2, P_3 P_4.

1. $N \rightarrow NuLuNrRdDdRrNuLuN$
2. $L \rightarrow LuNuLlDdRdDlLuNuL$
3. $R \rightarrow RdDdRrNuLuNrRdDdR$
4. $D \rightarrow DdRdDlLuNuLlDdRdD$

All right hand sides of the rules are again of the same length (it is a transformation of the original curve representation), but the number of non-terminals on the right hand side of the rules has increased from 4 to 9 compared to the Hilbert curve. The phase of agent creation has to take more steps (4 steps) and system will generate some spare agents (sixteen instead of nine). Also the phase of insertion of objects from right hand side of CF rules will take more steps (35 steps). The other phases of computation stays the same as in previous subsection.

4 Conclusion

In this contribution we showed APcol system with agent creation as a kind of P colonies is suitable model for turtle graphics commands generation. We presented the idea of construction of APcol system with capacity two that can compute turtle commands for given number of iteration as the input. We have described the construction of the system for two space-filling curves - Hilbert curve and Peano curve. For the construction, we exploited the existence of context-free rules to generate a string of directions representing these curves. Future work in the area of APcol systems ability research may focus not only on generating space-filling curves, but also on generation of command string of other curves that use parameters (for example a change of step length) or the stochastic command generation rules.

Notes and Comments. This work was supported partially by European Union under European Structural and Investment Funds Operational Programme Research, Development and Education project Zvýšení kvality vzdělávání na Slezské univerzitě v Opavě ve vazbě na potřeby Moravskoslezského kraje, CZ.02.2.69/0.0/0.0/18_058/0010238 and by the Silesian University in Opava under the Student Funding Scheme, project SGS/11/2019.

References

[1] Ceterchi, R., Subramanian, K.G. Generating pictures in string representation with P systems: the case of space-filling curves. J Membr Comput 2, 369–379 (2020).

[2] Cienciala, L., Ciencialová, L., Csuhaj-Varjú, E.: Towards P Colonies Processing Strings. In: Proc. BWMC 2014, Sevilla, 2014. pp. 102–118. Fénix Editora, Sevilla, Spain (2014)

[3] Cienciala, L., Ciencialová, L., Csuhaj-Varjú, E.: P colonies processing strings. Fundamenta Informaticae 134(1-2), 51–65 (2014)

[4] Cienciala, L., Ciencialová, L., Csuhaj-Varjú, E.: A Class of Restricted P Colonies with String Environment. Natural Computing 15(4), 541–549 (2016)

[5] Ciencialová, L. ,Csuhaj-Varjú, E., Cienciala, L.,Sosík, P.: P colonies. Bulletin of the International Membrane Computing Society 1(2):119–156 (2016).

[6] Csuhaj-Varjú, E., Kelemen, J., Păun, Gh., Dassow, J.(eds.): Grammar Systems: A Grammatical Approach to Distribution and Cooperation. Gordon and Breach Science Publishers, Inc., Newark, NJ, USA (1994)

[7] Hilbert, D. : Über die stetige Abbildung einer Linie auf ein Flächenstück. Math. Annln. 38 459–460 (1891).).

[8] Kelemenová, A.: P Colonies. Chapter 23.1, In: Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) The Oxford Handbook of Membrane Computing, pp. 584–593. Oxford University Press (2010)

[9] Kelemen, J., Kelemenová, A., Păun, G.: Preview of P Colonies: A Biochemically Inspired Computing Model. In: Workshop and Tutorial Proceedings. Ninth International Conference on the Simulation and Synthesis of Living Systems (Alife IX). pp. 82–86. Boston, Mass (2004)

[10] Kelemen, J., Kelemenová, A.: A Grammar-Theoretic Treatment of Multiagent Systems. Cybern. Syst. 23(6), 621–633 (1992),

[11] Păun, Gh., Rozenberg, G., Salomaa, A.(eds.): The Oxford Handbook of Membrane Computing. Oxford University Press, Inc., New York, NY, USA (2010)

[12] Peano, G.: Sur une courbe qui remplit toute une aire plane. Math. Annln. 36, 157–160 (1890).

[13] Prusinkiewicz, P., Lindenmayer, A.: The Algorithmic Beauty of Plants. Springer-Verlag, Berlin, Heidelberg (1996)

[14] Rozenberg, G., Salomaa, A.(eds.): Handbook of Formal Languages I-III. Springer Verlag., Berin-Heidelberg-New York (1997)

[15] Siromoney, R., Subramanian, K.G. : Space-filling curves and Infinite graphs. Lecture notes in Comp. Sci. 153 (1983) 380–391.