自己複製文字列と書き換え規則の共進化

杉浦 孔明 †, 鈴木 秀明 ‡, 塩瀬 隆之 †, 川上 浩司 †, 片井 修 †

† 京都大学大学院 情報学研究科
‡ ATR 人間情報科学研究所

Coevolution between Self-Replicating Strings and Rewriting Rule Sets

Komei Sugiura†, Hideaki Suzuki‡, Takayuki Shiose†,
Hiroshi Kawakami†, and Osamu Katai†

† Graduate School of Informatics, Kyoto University
‡ ATR Human Information Science Laboratories

Abstract: We develop a string rewriting system based on an Artificial Life (AL) system and investigate the automatic improvement of AL design. The system consists of character set and a set of string rewriting rules equivalent to Tierra instruction set. Each individual in the system is composed of a string as the genome and a rewriting rule set, and replicate its genome using its own rewriting rules. The system modifies both strings and rewriting rules by genetic operation. This paper presents several experimental results of coevolution among them.

Keywords: coevolution, string rewriting, artificial life, Tierra, regular expression

1. はじめに

人工生命研究は，生命の基本原理を探求するだけでなく，自然界の生命が持つ自律性や進化，適応能力といった優れた特徴を人工システムにおいて実現することを目指している．そのためには，構築しようとするシステムそのものを設計するのではなく，そのシステムが構築されるための環境を設計することが求められる．

人工生命システムの設計者が用意するものは，主に二つにわけることができる．すなわち，エージェント集団や環境を表すための**要素の集合**と，要素間の相互作用を規定する**反応規則の集合**である．要素として用いられるものには，記号 [9]，文字列 [6–8]，2 進文字列 [1]，λ 項 [2] などがある．また，反応規則は書き換え規則 [1,6–9]，λ-calculus [2] などで表される．

人工生命システムの進化性は，設計者が用意したこれらの基本デザインに大きく依存する．よって，変化に強くオープンエンドな進化の可能性を持つ，要素集合および反応規則を用意することが必要である．しかるに，これまでの人工生命の研究では，固定された反応規則のうえでエージェント集団を進化させるものが主であった．だが，人間の用意した規則は必ずしもオープンエンドな進化性を持つものであるとはいえない．

そこで本研究では，反応規則をシステム自身が改良するしくみを用意することによって，反応規則集合と要素集合を共に進化させる．そのために，人工生命システムを書き換え系によって表現し，文字列 (要素) だけでなく，書き換え規則 (反応規則) をも進化させる枠組みを実現する．

書き換え系を用いた書き換え規則最適化の研究例として Suzuki の研究が挙げられる [7]．また，Matsuzaki らは，人工生命システム Tierra [5] で用いられている機械語の命令セットを書き換え規則に翻訳する方法を提案している [4]．しかし，この提案にもかかわらず，Tierra のようなシステムを書き換え系によって実現し，書き換え規則を実際に進化させた研究は今までにない．そこで本研究では，Tierra においてエージェントと反応規則を同時に進化させるために，以下の二つのステップを踏む．まず，オリジナルの Tierra と論理的にほぼ等価なシステムを，文字列書き換え系によって構築する．次に，Tierra において固定されていた反応規則を，システム自身が改良するための枠組みを追加する．

本論文の構成は次の通りである．第 2 章では，オリジナルの Tierra について説明する．第 3 章では，本研究で作成したシステムについて述べる．第 4 章で，実験結果を報告し，それに対して考察を加える．最後に，第 5 章で結論を述べる．

2. Tierra とは

Ray による人工生態系 Tierra は，計算機の中でプログラムが進化することを示し，人工生命という研究パラダイムにひとつの契機を与えた [5]．Tierra はその設計の基本において，自然界における生命システムとの類推を用いている．

自然界の生態系において，生物は，主に太陽からのエネルギーを利用して物質を作り出し，資源をめぐる競争の中で淘汰される．この結果，最も効率よく自己複製できる遺伝形質が増えるように進化していく．これと同様にして，Tierra におけるディジタル生物 (エージェント) は，機械語で書かれた自己複製プログラムを遺伝形質とする．そのうえで，ディジタル生物は，CPU 時間とメモリ空間を求めて競争し，自身の遺伝形質をより効率よく複製できるように進化していく．

人工生命システムを構築するためには，システムに，突然変異や交叉といった遺伝操作に対するロバスト性を持たせなければならない．しかし，既存のプログラミング言語を用いたプログラムは変化に対して脆弱であるため，Tierra では，各個体のゲノムを表すために「Tierra

言語」を用いる．Tierra 言語とは，アセンブリ言語に似たものであり，命令の総数は 32 種類である．

各個体は，Tierra 言語のプログラム (機械語のシークエンス) で表されたゲノムを持ち，“生きているあいだ” ひとつの仮想プロセッサ (Tierra CPU) を与えられる．さらに，それぞれの Tierra CPU は 4 個のレジスタ (ax, bx, cx, dx) と容量 10 のスタックを持ち，ゲノムに書かれた機械語の命令を順に実行することで自身のゲノムを複製する．

3. 書き換え系による Tierra

3.1 Tierra との対応

本研究では，Tierra 言語で用いられる機械語を文字に対応させ，それに加えて，命令の実行を書き換え規則の適用によって表現する．本研究で作成したシステムでは，生物のゲノムを文字列で表現し，各々の生物に自らの持つ書き換え規則によって自己複製を行わせるようにする．さらに，書き換え規則に対して突然変異などの操作を加えることにより，書き換え規則を進化させることができる．

Table 1: Tierra と本システムの対応

	Tierra	本システム
要素集合	機械語	文字
生物のゲノム	機械語のシークエンス	文字列
反応規則	命令の実行	書き換え規則

3.2 要素集合

本研究では，書き換えの対象である要素集合として，以下の 5 種類の文字集合を用意する．

• 命令文字

命令文字は Tierra 言語の機械語を 32 種類の文字に対応させたものである．Table 2 に例を示す．

Table 2: 命令文字の例

命令文字	機械語	命令の意味
f	NOP0	no-operation
t	NOP1	no-operation
A	PUSHA	push ax on stack
B	PUSHB	push bx on stack
⋮	⋮	⋮
z	DIVIDE	cell division

• ポインタ文字

Tierra CPU は，次に実行する命令語のアドレスを指定するための命令ポインタ (IP) を持っている．これに対して本システムでは，IP を文字 ‘p’ で表し，p の直後にある命令文字が次に実行される命令であると解釈する．よって本システムでは，次に PUSHA という命令を実行する状態のゲノムは以下のように表される．

$$\cdots \texttt{pA} \cdots$$

このように，ポインタを文字で表現し，ゲノムの中に埋め込むことにより，ポインタの移動をゲノムの書き換えによって表現できるようになる．さらに本システムでは，生物の持つポインタの数をひとつに限定する必要はない．そのため，生物はゲノムの中に文字 p を複数持つことによって，命令を並列に実行することが可能になる．

• レジスタ文字，スタック文字，開始・終了文字

レジスタ文字は，Tierra CPU が持つ 4 個のレジスタ (ax, bx, cx, dx) をそれぞれ文字 ‘a’ から ‘d’ に対応させたものである．また，スタック文字は文字 ‘0’, ‘1’,. . . , ‘9’ で表され，これらは Tierra CPU の持つスタック (容量 10) に相当する．さらに，ゲノムの開始および終了位置を示す ‘[’ および ‘]’ をディジタル生物の “細胞膜” として用意する．

Tierra では，レジスタおよびスタックは仮想コンピュータにおけるメモリ上の命令語のアドレスを保持していた．これに対して本システムでは，レジスタやスタックは文字としてゲノムの中に埋め込まれている．これにより，レジスタやスタックを操作する命令を，ゲノムに対する書き換えとして表現することが可能になる．

本システムでは，レジスタやスタックが直接数を保存することはない．その代わりに，レジスタ文字やスタック文字は，ゲノムにおいて開始文字 ‘[’ からその文字までに含まれる，命令文字の数を保持するものと解釈する．ここで，次のようなゲノムの例を考える．

$$\texttt{[aABC}\cdots\texttt{]}$$

このとき，レジスタ文字 a は直後の命令文字 A の位置，つまり 0 を保持していることになる．ただし，レジスタ文字やスタック文字は細胞膜の外にでることはない．

Table 3: 要素集合

命令文字 (32 種)	t,f,A,...,Z,w,x,y,z
ポインタ文字 (1 種)	p
レジスタ文字 (4 種)	a, b, c, d
スタック文字 (10 種)	0, 1,..., 9
開始・終了文字 (2 種)	[,]

3.3 正規表現を用いた書き換え規則

本研究では，命令の実行を文字列の書き換えによって表現する．そのためには，命令の実行と等価な書き換え規則を用意しなければならない．そこで，各々の命令をひとつ又は複数の書き換え規則に翻訳する．以下では，Matsuzaki らの提案した書き換え規則 [4] との相違点に注意しながら，その特徴について説明する．

Matsuzaki らの提案した書き換え規則は独自の書式を用いているのに対し，本研究では，高度な記述力を持つ Perl の正規表現 [3] を利用して書き換え規則を作成した．このため，より複雑な書き換えを行うことができる．

それにくわえ，ひとつの書き換え規則を，正規表現のマッチングまたは置換の論理積として表現する．そのため，書き換え規則をひとつの正規表現の置換に対応させる場合に比べ，個々の正規表現を簡潔に書き表すことができる．よって，本研究で作成した書き換え規則は，いたずらに長く入り組んだ規則よりも，変化に対する許容性が高い．

さらに，Matsuzaki らの研究では，Tierra 言語の命令が複数の書き換え規則に対応することがあるが，それらは互いに独立ではないため，規則間に優先順位を与える必要があった．これに対し本研究では，Tierra 言語における 32 種類の命令を，互いに独立な 140 の書き換え規則として表現した．そのうえで，本システムでは，それらの規則を並べてひとつの規則集合を作り，規則を上から順に適用していくことで書き換えを行う．ただし，規則集合における規則の順序は並び替えが可能である．すなわち，本研究で作成した書き換え規則間には，固定された優先順位が存在しない．

• 命令と書き換え規則が 1 対 1 に対応する場合

いま，INCA(ax レジスタの値をインクリメント) が文字 'N' に対応するとすると，書き換え規則は以下のように表される[1]．

```
s/pN/Np/g && s/a($J)/$1a/g
```

ここに，

```
$N = [ポインタ文字，レジスタ文字，スタック文字]
$J = $N*[命令文字]$N*
```

とする．すなわち，$N はポインタ，レジスタ，スタックに対応する文字クラスを表し，これに*をつけることで 0 個以上の任意個の$N を表す．したがって，$J は前後に 0 個以上の$N を持った，任意の命令文字ひとつにマッチする．

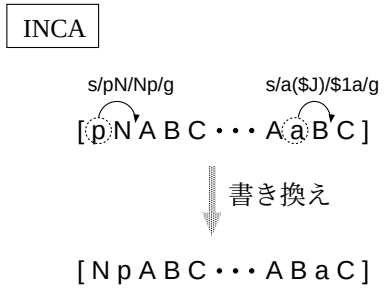

Figure 1: INCA の実行

INCA に対応する書き換えを行う様子を Figure 1 に示す．まず，前半部の置換によって，INCA を指していたポインタを 1 つ右へ移動させる．次に，後半部では，a の後ろにある命令文字 1 個 (レジスタ文字などを含むこともある) の位置と，a の位置を交換する．こうすることにより，開始文字 '[' から a までの間にある命令文字の数は 1 だけ増加する．3.2 で述べたように，これはレジスタの値を 1 だけ増加させたことを意味する．

[1] s/PATTERN/REPLACEMENT/g は，対象文字列中で PATTERN にマッチ可能な全ての部分を REPLACEMENT で置換する．&&でつながれた複数の置換は左から順に施され，照合しないものがでてきた時点で打ち切られる．

• 命令と書き換え規則が 1 対 n に対応する場合

ひとつの命令に対して複数の書き換え規則が対応する例として IFZ(cx レジスタの値が 0 ならば次の命令を実行し，そうでなければスキップする) が挙げられる．いま，IFZ を文字 'U' に対応させたとすると，IFZ と等価な書き換え規則は以下の 2 つになる[2]．

```
    /\[$N*c/   &&   s/pU/Up/g
!   /\[$N*c/   &&   s/pU($J)/U$1p/g
```

Figure 2 に，これらを用いた書き換えが行われる様子を示す．この例では，はじめの/\[$N*c/によって，ある配列が対象文字列の中で検索され，その有無によって右の置換ルールの実行が制御される．これはゲノムにある "調節配列" の働きにたとえることができる．

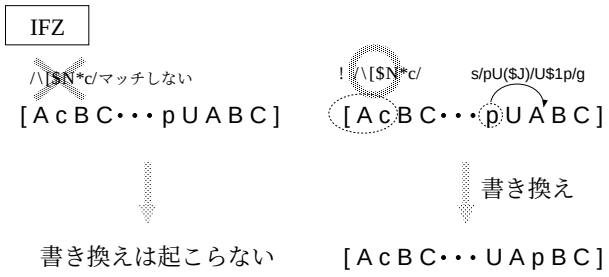

Figure 2: IFZ の実行

3.4 書き換え規則の並び替え

本研究では，個々の生物がそれぞれ独自の書き換え規則集合 **R** を持ち，**R** を用いて自己複製を行う．ただし，生物はタイムステップごとに **R** に含まれる全ての規則を上から順に実行するとする．よって，1 タイムステップにおける適用回数は規則の並びに依存する．たとえ同じゲノムを持つ生物同士でも，**R** の中の規則の並びがゲノムに適しており，規則の平均適用回数が大きいほど，少ないステップで自己複製を行うことができる．

そこで，本研究では，進化を用いてより優れた規則集合を得るために，規則集合に対する選択を導入する．ここに，個々の書き換え規則の適合度を，その規則の適用回数と定義する．書き換え規則集合 **R** における並び替えの手順は以下の通りである (Figure 3)．

1. 規則を適用回数にしたがってソートする
2. 下から l 個の規則を削除する
3. 上位 l 個の規則を選び，交叉や突然変異などの操作を加えて，新たに l 個の規則を生成する
4. 新規則を **R** の中央に挿入する

以上の操作は，タイムステップごとに確率 p で行われる．この操作によって，適用回数の少ない (必要性の薄い) 規則は排除され，必要性の高い規則は選択される．

[2] /PATTERN/は，対象文字列に対してパターンマッチを行い，マッチすれば真を返す．!は論理否定を表す．

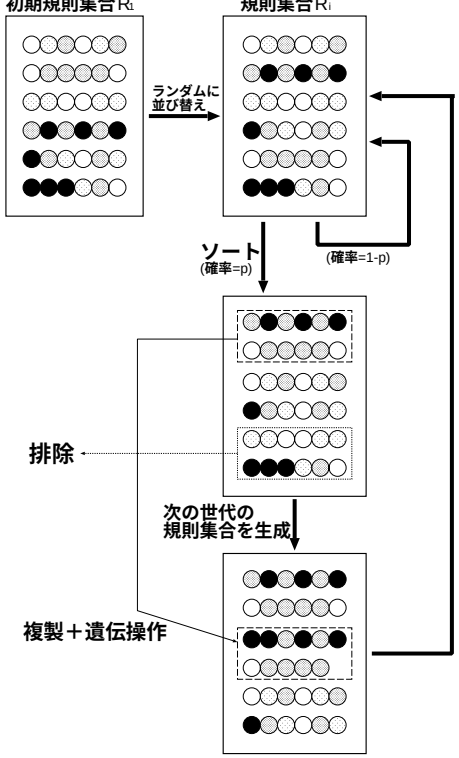

Figure 3: 規則の並び替え

3.5 突然変異

Tierra において，突然変異はゲノムに対してのみ行われていた．しかし本システムでは，変異させる対象は，a) ゲノム，b) 書き換え規則，の 2 つである．以下では，それぞれの場合について述べる．

• ゲノムに対する突然変異

本システムでは，生物のゲノムは文字列で表されるため，Tierra で用いられていた置換以外の突然変異を導入することができる．そこで，本システムでは 1) 挿入，2) 削除，3) 置換，の 3 種類の突然変異を用意する (ただし，置換は挿入と削除を組み合わせて表現することもできる)．

• 書き換え規則に対する突然変異

書き換え規則に対する突然変異は，1) 重複，2) 削除，3) 正規表現の変化，の 3 種類である．いま，書き換え規則 R を

$$R = r_1 \ \& \ r_2 \ \& \cdots \& \ r_i \ \& \cdots \& \ r_n$$

とすると，上の 3 種類の操作は以下のように表される．

$$\begin{aligned}
\text{重複:} \quad & r_1 \ \& \ r_2 \ \& \cdots \& \ \mathbf{r}_i \ \& \cdots \& \ r_n \\
& \Longrightarrow r_1 \ \& \ r_2 \ \& \cdots \& \ \mathbf{r}_i \ \& \ \mathbf{r}_i \ \& \cdots \& \ r_n \\
\text{削除:} \quad & r_1 \ \& \ r_2 \ \& \cdots \& \ \mathbf{r}_{r-1} \ \& \ \mathbf{r}_i \ \& \ \mathbf{r}_{i+1} \ \& \cdots \& \ r_n \\
& \Longrightarrow r_1 \ \& \ r_2 \ \& \cdots \& \ \mathbf{r}_{r-1} \ \& \ \mathbf{r}_{i+1} \ \& \cdots \& \ r_n \\
\text{変化:} \quad & r_1 \ \& \ r_2 \ \& \cdots \& \ \mathbf{r}_i \ \& \cdots \& \ r_n \\
& \Longrightarrow r_1 \ \& \ r_2 \ \& \cdots \& \ \mathbf{r}'_i \ \& \cdots \& \ r_n
\end{aligned}$$

r_i を r'_i に変化させる方法として，ゲノムの突然変異と同様に，1) 挿入，2) 削除，3) 置換，の 3 種類を用意した．ただし，正規表現の置換を変化させる場合は，その規則が実際に適用されたとしても，書き換え前後で命令文字の数が変わらない (リソース保存する) ように正規表現を変化させる (一部例外もある)．

3.6 淘汰

Tierra では，80 語の命令でできたゲノムを持つ祖先種が，メモリ上で自己複製を繰り返す．このとき，生物はメモリ (大きさ固定) に占める割合が一定以下になるように管理されている．これに対し，本システムはゲノムを文字列で表しているため，ゲノムの長さを伸縮させることが容易である．そこで，生物が際限なく増殖するのを避けるために，個体数に上限 N_{max} を設ける．総個体数が N_{max} を超えた場合は，集団からランダムに個体を選んで削除する．

さらに，自己複製能力のない個体を排除する方法として，書き換え規則の適用回数による淘汰を行う．いま，ある個体が，ゲノムに不利な変化を起こしたために，書き換え規則を適用できなくなったとする．このような個体を排除するために，タイムステップごとに規則の適用回数を調べて，書き換えが 1 回も起こらなかったものを集団から排除する．

本システムでは，祖先種は文字列でできたゲノムを持ち，Tierra の命令セット (祖先規則集合) を用いて自己複製を行う．ここで，この祖先種のゲノムは，Tierra の祖先種のゲノムを文字列に翻訳したものを用いる．このような祖先種を N_{max} 個集め，それぞれに 140 個の書き換え規則をランダムに並べ替えたものを与える．

以上のような枠組みを用意したうえで，個体と規則を共に進化させて，よりよい規則集合を得るために，本システムを用いて実験を行った．

4. 実験結果および考察

実験では，以下のパラメータを用いている．

$$\begin{aligned}
N_{max} &= \text{最大個体数} \\
m_g &= \text{ゲノムに対する突然変異率} \\
m_r &= \text{規則に対する突然変異率} \\
p &= 1 \text{ タイムステップで並び替えが行われる確率} \\
l &= 1 \text{ 回の並び替えで差し換えられる規則の数}
\end{aligned}$$

これらの値は予備実験の結果をもとに，$N_{max} = 32$，$m_g = 0.01$，$m_r = 0.9$，$p = 0.02$，$l = 14$ のように決定した．

実験は 2.4GHz CPU の計算機サーバーを用いて行い，10 万タイムステップの実行に約 4 時間かかった．

Figure 4 は，1 タイムステップにおいて適用された規則の数を，時間に対してプロットしたものである．また，Figure 5 は，細胞分裂のインターバルをタイムステップに対してプロットしたものである．ここで，細胞分裂のインターバルとは，前々回の細胞分裂から前回の細胞分裂までにかかったステップ数を意味する．

Figure 4 より，規則の適用回数の最大値および平均値は，上下はするものの，増加していることがわかる．こ

れは，規則集合から不要な規則を排除して，有用な規則を獲得したためであると考えられる．また，Figure 5 より，平均的な分裂までのインターバルは 1000 ステップにおいて約 500 だったのに対して，10 万ステップでは約 30 へと改善されている．以上の結果から，10 万タイムステップ後の生物は，祖先種よりよい規則集合を獲得したと考えられる．

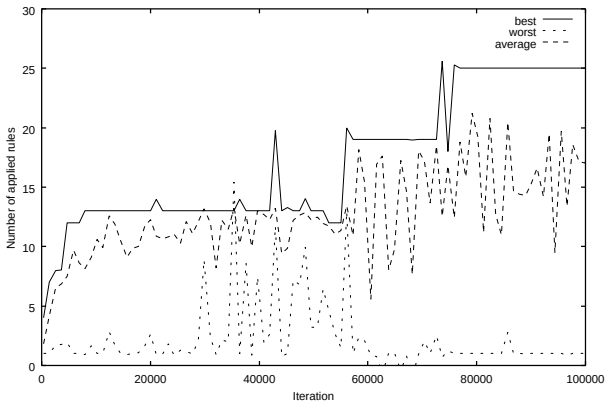

Figure 4: 規則の適用回数の変化

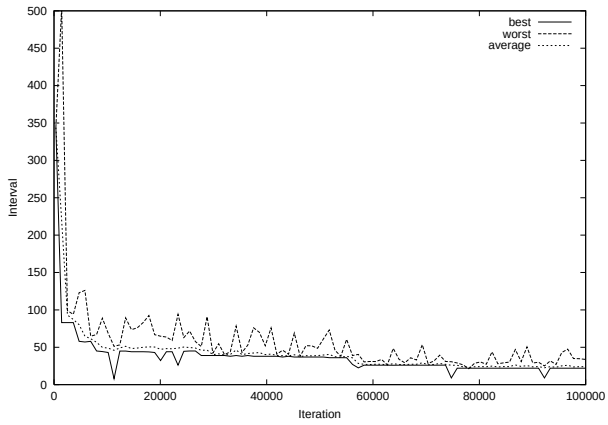

Figure 5: 分裂までのインターバルの変化

また，分裂した直後のゲノムの長さは祖先種において 87 であったものが，10 万ステップ後の子孫種では 69 に短くなっていた．ゲノムが短いほど分裂を早く行えるため，ゲノムについても改良されたといえる．

以上のことから，エージェントと書き換え規則を共に進化させることができたと考えられる．

5. おわりに

人工生命システム Tierra をもとにした書き換え系を構築した．まず，Tierra 言語における 32 種類の命令を，互いに独立な 140 の書き換え規則に翻訳した．高度な記述力と，規則自身への変化に対する高い許容性を同時に実現するために，それらの書き換え規則を，それぞれ正規表現のマッチングまたは置換を論理積演算子でつないだものとして表現した．また，作成した書き換え系に，書き換え規則に対する遺伝的操作を導入することによって，人手で作成された書き換え規則をシステムが改良するしくみを用意した．本システムを用いて実験を行なった結果，文字列と書き換え規則の共進化が起こることが示された．

今後の課題として以下のことが挙げられる．

- 進化によって得られた規則集合の解析と，その進化性を評価する
- モデルに環境を導入して，リソース保存型の人工生命システムを構築する
- モデルに対して多細胞性や摂取・排泄を導入する

謝辞

本研究を進めるにあたり，有意義な議論をして頂いた会津大学 大学院博士課程の松崎 周一さんに感謝致します．

参考文献

[1] P. Dittrich and W. Banzhaf. Self-evolution in a constructive binary string system. *Artificial Life*, Vol. 4, No. 2, pp. 203–220, 1998.

[2] W. Fontana. Algorithmic chemistry. In C. G. Langton, C. Taylor, J. D. Farmer, and S Rasmussen, editors, *Artificial Life II: Proceedings of the Second Artificial Life Workshop*, pp. 159–209. Addison-Wesley, 1992.

[3] J. E. F. Friedl. *Mastering Regular Expressions*. O'reilly, 1997.

[4] S. Matsuzaki, H. Suzuki, and M. Osano. Tierra instructions implemented using string rewriting rules. In *Proceedings of the Fifth International Conference on Humans and Computers (HC-2002)*, pp. 167–172, 2002.

[5] T. S. Ray. An approach to the synthesis of life. In C. G. Langton, C. Taylor, J. D. Farmer, and S Rasmussen, editors, *Artificial Life II: Proceedings of the Second Artificial Life Workshop*, pp. 371–408. Addison-Wesley, 1992.

[6] H. Suzuki. Evolution of self-reproducing programs in a core propelled by parallel protein execution. *Artificial Life*, Vol. 6, No. 2, pp. 103–108, 2000.

[7] H. Suzuki. String rewriting grammar optimized using an evolvability measure. In J. Kelemen and P. Sosik, editors, *Advances in Artificial Life (6th European Conference on Artificial Life Proceedings)*, pp. 458–468. Springer-Verlag, Berlin, 2001.

[8] H. Suzuki and N. Ono. Universal replication in a string rewriting system. In *Proceedings of the Fifth International Conference on Humans and Computers (HC-2002)*, pp. 179–184, 2002.

[9] Y. Suzuki and H. Tanaka. Chemical evolution among artificial proto-cells. In M. A. Bedau, J. S. McCaskill, N. H. Packard, and S. Rasmussen, editors, *Artificial Life VII: Proceedings of the Seventh International Conference on Artificial Life*, pp. 54–63. MIT Press, 2000.