文字列のもつ帰納的構造の代数的性質に関する考察

A Note on the Property of Strings from the Viewpoint of Algebra

坂本量平† 野村亮†
Ryohei SAKAMOTO† Ryo NOMURA†

†専修大学 ネットワーク情報学部

要旨:

計算機科学における主たる研究対象の一つに文字列がある．他方，代数の文脈では文字列は研究対象ではあるが中心的ではない．両者の研究は視点が異なっているために，同じ研究対象であっても，互いの結果は共有されていないようである．本稿では文字列について両者の接続を次のように試みる．(1) 計算機科学の視点から文字列を帰納的に定義する．(2) さらに帰納的に定義された文字列を代数的に分析するために始代数の定義を用いて定式化する．(3) 最後に始代数としての文字列が自由モノイドの性質を持つことを確認する．このように文字列が自由モノイドであることを確認することで代数の視点から文字列を考察することが可能になる．その一つの帰結として本研究では代数における自由モノイドと自由群の関係を用いて「符号付き文字列」を提案する．両者の接続の可能性を示す一つの例になると考えられる．

Abstract:

In the field of the computer science, to investigate the property of strings is one of main objects such as the regular expression. On the other hand, in the field of algebra the property of strings is also investigated from the different viewpoint with the computer science. In this study we first define the string inductively from the viewpoint of the computer science and formalize it from the viewpoint of algebra by using the notion of initial algebra. Then, we confirm that the set of all strings has the property of free monoid. Finally, we propose a new notion called “signed string” as a consequence of this consideration.

1. はじめに

計算機科学と代数は文字列を研究対象として共有する．しかし，それぞれの視点から映る光景は異なっている．

計算機科学にとって文字列は主たる研究対象の一つである．例えば，オートマトンやチューリングマシンといった計算機械を用いて文字列の持つ性質について多くの研究がされた．これらの研究は文字列の持つ帰納的な構造に着目している．

一方，文字列を代数系としてとらえる視点も存在する．連接（concatenation）という演算によって文字列全体はモノイド（monoid）と呼ばれる代数系に分類される．加えて，文字列全体がなすモノイドは自由（free）という性質を持つ．自由とは，端的に言えば，基底（base）を持つ代数系である．文字列については文字全体の集合が基底に相当する．

上記のように，文字列は，計算機科学では帰納的な構造として，代数では自由代数として研究されてきたが，互いの結果はそれほど接続されていない．これは帰納的な構造として定義された文字列が代数の議論になじまないからであろう．そこで本稿では始代数（initial algebra）という概念を導入することによりこれらの接続を試みる．具体的には，まず文字列を帰納的に定義する．さらに帰納的な構造として定義された文字列を始代数として定式化する．最後に，始代数として定式化される文字列が自由性を持つことを示す．

上記のように文字列の帰納的な構造を代数の視点から考えることはそれ自身が大変興味深いというだけでなく，新たな概念を考える指標となる．

文字列は代数においてモノイドに分類される．モノイドに演算法則すなわち，逆元の存在を加えると群（group）と呼ばれる代数系になる．代数の世界では，モノイドから群を考えることが極めて自然な流れである．同様に自由モノイドとしての性質を持つ文字列に対して自由群（free group）を考えることは自然であろう．本研究では自由群という代数的性質を持った帰納的な構造として，符号付き文字列（signed string）を導入する．

この考え方は自然数から整数への拡張のアナロジーとして考えることができる．自然数に符号（正と負）を導入することで整数が得られる．自然数の和演算における逆元とはすなわち負の数に他ならず，負の数を含まない自然数上では引き算を行えない場合がある．例えば，自然数の世界では 4 から 7 を引くことができない．そこで逆元を加えた集合，すなわち整数，を考えることは極めて自然であろう．この発展は代数の語彙を借りれば，モノイドから群への発展である．

私たちは同様の試みを文字列について考えたい．文字列においては連接が自然数における和演算に相当する．では逆元は何であろうか．私たちは負の文字という概念を知らない．そのため，文字列から文字列の減算を定義できない場合がある．例えば，*abcdef* という文字列を考えよう．この文字列から *def* という文字列を減ずると *abc* が得られる．では，*ghi* という文字列を減ずると何が得られるだろうか．我々はその答えを知らない．自然数の範囲で 4 から 7 が引けないように，私たちは文字列を自由に減ずることができない．この例は，計算機科学で多く語られてきた文字列に代数的な視点を入れることで発展の余地があることを示唆している．代数と計算機科学の接続により，自然数が整数に発展したように，文字列を符号付き文字列に発展させることができる．

さて，以上の視座から本稿の仕事は次のように導かれるだろう．まず，2 章にて数学的な準備，モノイドと群，始代数，自由代数といった概念を説明する．そして 3 章で文字列を始代数として定義し自由モノイドであること示す．最後に 4 章で符号付き文字列を始代数として新しく導入し，これが自由群であることを示す．さらに 5 章で符号付き文字列を実装す

る．なお，本論文は卒業論文（文献[1]）をまとめたものであり，形式的な証明など，厳密な議論はそちらに譲る．

2.　準備

2.1. モノイドと群

モノイドとはある集合とその集合上で単位元と結合的な二項演算（乗法と呼ばれる．ただし，乗法とは必ずしも積演算を意味しない）を持つ代数である．元の全体を M，乗法演算子を・，単位元を e で表そう．e が単位元であるとは，任意の M の元 x について $x \cdot e = x = e \cdot x$ が成り立つことを意味する．また，乗法が結合的とは，任意の M の元 x,y,z について，$x \cdot (y \cdot z) = (x \cdot y) \cdot z$ が成り立つことを意味する．

例 2.1.1（自然数）

自然数はモノイドの典型例である．自然数全体の集合 $\boldsymbol{N}$ と加法＋，ゼロ 0 はモノイドをなす．ただし，ここでは 0 を自然数に含める（文献[3]など）．

例 2.1.2.（文字列）

文字列もモノイドである．文字の集合を Σ として，Σ に属す文字からなる文字列全体の集合 Σ^* とする．Σ^* は連接と空列 ε によってモノイドになる．

群とは，任意の元について逆元が存在するモノイドである．元全体の集合を G，乗法演算子を・，単位元を e としよう．元 x の逆元 x^{-1} とは，$x \cdot x^{-1} = e = x^{-1} \cdot x$ が成り立つ元である．たとえば，自然数全体の集合 $\boldsymbol{N}$ は群ではない．0 については逆元が存在する．0 自身である．しかしながら，1 についてその逆元は存在しない．つまり，$1 + x = 0 = x + 1$ を満たす自然数 x は存在しない．同様に文字列 Σ^* も群ではないことがわかる．一方，整数に対して和演算を考えたときは，任意の整数 x について，$-x$ が x の逆元であることより群をなすことがわかる．

2.2. 逆演算を用いた群の定義

自然数と整数はともに和演算を持つが，その逆演算である減算は計算可能な範囲が異なる．自然数においては引き算ができないときがある（4 から 7 を引けないように）．しかし，整数の範囲ではすべての範囲で引き算ができる（4 から 7 を引くと −3 である）．

これは群の定義を逆元ではなく，乗法の逆演算である除法を用いて再定義できることを示唆している．

代数の議論では群の定義に除法が登場することは稀である．除法は逆元によって代替可能だからである．また，除法が演算として乗法のようによい振る舞いをしないからである．たとえば，結合的ではない．しかし，計算機上で群を与えたいときは，逆元はなじまない．逆元を自然に定義しようとすれば，定義に除法が現れるからである．

たとえば，数 x の逆数（積演算についての逆元）を考えよう．逆数は x^{-1} もしくは $1/x$ で表現できる．前者は単項演算の適用により逆数を与えているが，後者は 1 から x を割っている（二項演算である）．同様に数 x の符号を逆にした数 $-x$ も $0 - x$ によって表現できる．前者は単項マイナス演算子によって，後者は減法（二項演算）によって表現している．

このように，逆元は 2 つの方法――単項演算による方法と二項演算（逆演算）による方法――によって，定義ができる．

しかし，通常の代数の議論では前者の方法しか登場しない．これも，代数の議論に計算の視点が注がれていないことを示す証拠になるだろう．

私たちは「符号付き文字列」の導入においても，逆元からは出発しない．連接の逆演算である逆連接を導入する．これは $abcdef / def = abc$ となるような演算である．

文献[1]では，群を逆元からではなく除法から再定義している．

2.3. 準同型について

代数を議論する上で準同型（homomorhism）は重要である．準同型とは代数の構造を保存した写像であり，線形代数における線形写像，順序集合における単調写像，位相空間における連続写像などは準同型写像である．

2 つのモノイド M，N が与えられたとき，写像 $\phi : M \to N$ が準同型であるとは，次の 2 つの等式を満たすことである．

$\phi(e) = e$，

$\phi(x \cdot y) = \phi(x) \cdot \phi(y)$.

単位元を単位元に写す写像であり，また乗法によって結ばれた 2 つの項がその終域においても，結合が維持されている写像である．これをモノイド準同型と呼ぶ．準同型は具体的な計算において効果を発揮する．たとえば，あるモノイドの元を準同型写像で写すとき，その元がいくつかの元によって展開可能であるとしよう．モノイドの元全体は無限に存在したとしても，すべての元を有限の元の展開によって表現できるとすれば，すべての元に対して対応が可能になる．

これは準同型を表現（representation）できることを示唆している．たとえば，線形写像は行列で表現することができる．指数関数を思い出そう．指数関数は指数法則を満たす関数のことであるが，指数関数は底（base）によって表現できる．指数には無限の数が乗るが，それらすべての割り当て先を一つの底のみによって計算できる．これは指数関数（のみならずモノイド準同型）が 1 つの項で表現できることを意味している．

群についても同様である．群の場合は逆元についても保存しなければならない．つまり，上で挙げた条件に次が加えられる．

$\phi(x^{-1}) = \phi(x)^{-1}$.

以上の性質は自由代数において重要になる．

2.4. 帰納的定義から始代数へ

始代数についての議論は主に文献[2]を参考にしている．詳しくはそちらを参照されたい．

自然数や文字列は帰納的（inductive）な構造を持っている．

自然数の定義を思い出そう．ただしここでは 0 を自然数に含めることとする．

1.　0 は自然数である．
2.　n が自然数のとき，σn は自然数である．
3.　自然数はこのようなものに限る．

この規則によって自然数は 0，$\sigma 0$，$\sigma \sigma 0$，$\sigma \sigma \sigma 0, \ldots$ と帰納的に生成される．ただし σ は自然数において後者（successor）を指示する構成子（constructor）である．つまり，

σ 0 は 0 の次の数である 1 を指し，σ σ 0 は 2 を指す．また，最後の条件によって，帰納的生成で得られないものは自然数ではないことが保証される．

参考 2.4.1.（プログラミングにおける自然数）

プログラミングの文脈において，帰納的な定義は重要である．リストや木などのデータ構造は帰納的な構造を持っていることが多い．再帰的データ型との関係も深い．たとえば，自然数の型 Nat を次のように再帰的データ型として定義できる．記法については文献[2]を参考．

Nat ::= Zero | Succ Nat.

これは，先に行った帰納的定義と本質的に同じある．

さて，帰納的な構造を持つと，関数を帰納的に定義することができる．たとえば，自然数 *n* に対して *n* 番目の偶数を返す関数 *f* を定義しよう．次の 2 つの等式で十分である．

$f(0) = 0,\ f(\sigma n) = f(n) + 2.$

たとえば，*f*(3)は次のように計算される．

$f(3) = f(2) + 2 = f(1) + 2 + 2 = f(0) + 2 + 2 + 2 = 0 + 2 + 2 + 2 = 6.$

これは関数の帰納的定義と呼ばれる．

始代数（initial algebra）は関数の帰納的定義を一般化したものである．詳しくは文献[2]に譲るが，いくつかの等式を与えると関数が一意に定まるという条件によって集合を定義する．これが始代数の発想である．

始代数の方法を使って，自然数を定義しよう（文献[2]の Nat 型を参考）．自然数の集合 ***N*** を次の条件を満たす集合として定義する．任意の集合 X，関数 $h : X \to X$，*X* の元 *c* に対して，$f(0) = c, f(\sigma n) = h(f(n))$ を満たす関数 *f* が一意に定まる．

始代数とは，いくつかの等式を与えると，始代数を始域とする関数が一意に定まるような集合もしくは型である．帰納的な構造はすべてこの始代数として表現ができる．たとえば，文字列，リスト，木，正規表現などである．

始代数の有効性は関数の定義にある．自然数については加法や指数関数など，自然数を始域とする関数を帰納的に定義することができる．

自然数は無限に存在する．無限に存在するにもかかわらず，有限の規則だけから任意の自然数に対して，計算することが可能になる．実数ではそうはいかない．有限の規則から無限の結果を得る．これが計算の重要な視座である．始代数はこれを数学的に表現することに成功している．

2.5. 自由代数

ここでは，自由代数の定義は文献[3]，[8]を元にしている．厳密な定義はそちらに譲る．

Jakob Nielsen は群論に「自由（free）」という概念を導入した（文献[7]）．ある代数系が自由であるとは，独立生成系（independent generator）を持つことを意味する．独立生成系は基底（base）とも呼ばれる．つまり，基底を持つ代数系は自由である．

線形代数を思い出そう．あるベクトル空間のベクトルをいくつか集めたとき，それらについて，線形独立，線形従属，張る，基底であるといった性質が議論される．通常，ベクトル空間には必ず基底が存在する．そのため，ベクトル空間は自由である．

ある代数系が自由であることを示すためには，基底の存在を示せば十分である．ある代数系から基底を発見する．この方向とは逆に，ある集合を基底として代数系を構成することがある．これは自由生成（free generate）と呼ばれる．また，このようにして生成された代数系を自由代数（free algebra）と呼ぶ．

例 2.5.1.（文字と文字列）

文字の集合Σから文字列の集合 Σ^*を生成することができる．これは自由生成の典型例である．Σを基底とするモノイド Σ^*が自由生成されていることになる．

例 2.5.2.（冪集合）

冪集合（powerset）も自由生成の例である．ある集合 *X* が与えられたとき，*X* の部分集合全体を *X* の冪集合 *P*(*X*)と表す．*X* の部分集合同士を合併（union）することができる．この演算によって冪集合は冪等可換モノイドである．

基底と自由代数は拡張（extension）によって関係が結ばれている．つまり，自然な定義によって，基底からある代数系への関数について，関数の始域を基底から自由代数へ拡張できる．ただし，拡張された関数は準同型として定義する．

例 2.5.3.（文字から文字列への拡張）

文字の集合Σからあるモノイド *M* への関数 $f : \Sigma \to M$ を考えよう．この関数はΣに属す各文字に対してモノイド *M* のある元を返す．*f* の始域はΣである．これを Σ^*に拡張できる．つまり，文字列に対してモノイド *M* の元を割り当てる関数に拡張する．たとえば，文字列 *abc* に対して *f* は $f(abc) = f(a) \cdot f(b) \cdot f(c)$を割り当てる．ただし，ここで右辺に現れる演算子・は *M* 上の乗法である．

基底の存在は，拡張の一意性によって言い換えられる．拡張によって得られた準同型は，基底の割り当てを保存しなければならない．これは基底から自由代数への挿入関数（insertion）との可換性により定義できる．

例 2.5.4（文字から文字列への挿入関数）

文字の集合Σに属す文字について，長さ 1 の文字列を返す関数を $\eta : \Sigma \to \Sigma^*$とする．このとき，拡張は任意のΣに属す文字 *a* について次の等式を満たす（可換である）．

$f^*(\eta(a)) = f(a).$

2.6. 考察（始代数と自由代数）

計算機科学と代数の対比は始代数と自由代数の対比に重ねられる．両者は文字列において交差する．計算機科学の方法では文字列は始代数として，代数の方法では文字列は自由代数として定義される．そして，後に示すように，帰納的に定義される文字列（始代数）は自由モノイド（自由代数）である．

帰納的定義または始代数は有限の等式から関数の定義（無限の要素についての割り当て）を導いた．自由代数も共通している．関数の始域を基底から自由代数へ拡張しているから

である．基底が有限集合であっても，自由代数の元は無限に存在する．拡張は関数の帰納的定義とこのような共通性を持つ．

始代数によって得られる対象と自由代数によって得られる対象は一部共通しているとはいえ（自然数や文字列など），異なる部分の方が大きいだろう．たとえば，ベクトル空間や正規表現は異なる世界に属している．

始代数は帰納的な定義を一般化したものであるため，構成方法を具体的に提示してくれる．とはいえ，構成された対象が代数的な構造を持っているとは限らない．自然数や文字列の代数的な構造は加法もしくは連接に現れているが，帰納的な定義において登場するのは後者関数といった構成子であって，代数構造を持ったのは偶然である．

自由代数の場合は，自由生成の具体的な方法を提示していない．自由生成された結果得られる自由代数の性質について議論しているに過ぎず，それをどのようにして，計算機上で構成するかについて関心がない．そのため，自由モノイドが帰納的に定義できることも偶然である．

3.　文字列について

以上の数学的準備の元で文字列の代数的性質を考察する．まず，文字列を帰納的に定義し，それを始代数として定式化する．この段階では計算機科学の領域を出ていない．目指されるのは，代数の領域との接続である．それは帰納的に定義した文字列が自由モノイドであることの証明によって達成される．詳しくは文献[1]を参照されたい．始代数として得られた文字列の集合に代数的な構造を与えるために，連接を定義する．これは関数の帰納的定義により可能である．また，自由であることを示すために拡張を定義する．得られた諸演算がモノイドの条件を満たすこと，拡張の条件を満たすことを示せば，目標は達成される．

3.1 文字列の始代数性

まず，文字列を帰納的に定義しよう（文字列の定義は文献[2]における List の定義を元にしている）．文字の集合を Σ として，文字列の集合 Σ^* を 2.5 節における自然数の定義と同様に次のように定義する．

1. ε は Σ^* の要素である――ε は空列と呼ばれる．
2. α は Σ^* の要素，a は Σ の要素であるとき，$\alpha{:}a$ は Σ^* の要素である．
3. Σ^* の要素は次のようなものに限る．

Σ の要素（文字）をローマ字 $a,b,c,\cdots$ で表し，Σ^* の要素（文字列）をギリシア文字 $\alpha,\beta,\gamma,\cdots$ で表そう．構成子 : によって文字列の後ろに文字が追加される．この構成子は Cons と呼ばれる（文献[2]など）．関数プログラミング Lisp が由来，constructor の略形である．

この構成方法により，ε，$\varepsilon{:}a$，$\varepsilon{:}a{:}b$，$\varepsilon{:}a{:}b{:}c$，…と文字列が帰納的に生成される．ただし $\varepsilon{:}a{:}b{:}c$ は文字列 abc を意味する．

さて，2.4 節で帰納的に定義された自然数を始代数を用いて定義したように，帰納的に構成された集合 Σ^* は始代数により定式化できる．集合 Σ^* は次のような集合である．任意の集合 X，関数 $h: X \times \Sigma \to X$，X の元 c に対して，$f(\varepsilon) = c$，$f(\alpha{:}a) = h(f(\alpha), a)$ を満たす関数 $f: \Sigma^* \to X$ が一意に定まる．

たとえば，関数 h と定数 c が与えられたとき，$f(ab)$ は次のように計算される．$f(ab) = h(h(c, a), b)$.

例 3.1.1.（関数 length）

文字列の長さ（length）を返す関数を帰納的に定義してみよう．$length: \Sigma^* \to \boldsymbol{N}$ を次の 2 つの等式で定義する．

$length(\varepsilon) = 0,\ length(\alpha{:}a) = length(\alpha) + 1.$

このように，始代数によって文字列の集合を定式化することで，文字列の集合を始域とする関数を帰納的に定義できる．

3.2 諸演算の定義

文字列の集合 Σ^* を始代数によって定義をした．しかし，まだ，Σ^* がモノイドであることは明らかではない．モノイドであるためには乗法を定義しなければならない．Σ^* における乗法とは文字列と文字列の連接（concatenation）である．たとえば，文字列 abc と def を連接すると $abcdef$ になる．また，拡張も定義しなければならない．

いずれも，始代数によって得られた関数の帰納的定義によって定義可能である．

連接を表す演算子を・として，・を次のように帰納的に定義する．

$\alpha \cdot \varepsilon = \alpha$，　$\alpha \cdot (\beta{:}a) = (\alpha \cdot \beta){:}a.$

続いて，拡張を定義しよう．任意のモノイド M と関数 $f: \Sigma \to M$ が与えられたとき，拡張された関数 $f^*: \Sigma^* \to M$ を次のように定義する．

$f^*(\varepsilon) = e,\ f(\alpha{:}a) = f^*(\alpha) \cdot f(a).$

補足しよう．e はモノイド M の単位元である．また，2 つめの等式の右辺に現れている・はモノイド M の乗法演算子である．以上によって関数 $f: \Sigma \to M$ は $f^*: \Sigma^* \to M$ に拡張された．

例 3.2.1（関数 length）

文字列の長さを返す関数は拡張によって定義できる．拡張される関数は，任意の文字について 1 を返す定数関数 $f: \Sigma \to \boldsymbol{N}$ である．自然数の集合は 0 を単位元，＋を乗法とするモノイドだった．よって，拡張によって得られる関数，$f^*: \Sigma^* \to \boldsymbol{N}$ は次の等式を満たす関数である．

$f^*(\varepsilon) = 0,\ f^*(\alpha{:}a) = f^*(\alpha) + 1.$

これは $length$ の定義と等しい．

さらに，基底の存在（Σ は Σ^* の基底である）を示すために，各文字 a に対して長さ 1 の文字列 a を返す関数も必要である．これは挿入（insertion）と呼ばれる．$\eta: \Sigma \to \Sigma^*$. これは帰納的に定義する必要はなく，$\eta(a) = \varepsilon{:}a$ として定義される．挿入はプログラミングにおける型変換と類似している．

3.3 演算の性質

挿入，連接，拡張を定義した．これらは文字列の集合 Σ^* が自由モノイドであることを示すために要請された．自由モノイドであるためには，いくつかの条件（連接演算が結合的であること，空列が連接について単位元であること，拡張がモノイド準同型であること，拡張が一意的であること）を示さなければならない．それらを示すために演算の性質を調べ

ておこう．ただし，詳しい証明は文献[1]に譲る．

Cons，連接，挿入の間には次の等式が成り立つ．

$\alpha : a = \alpha \cdot \eta(a)$.

文字を追加する操作は長さ1の文字列を連接する操作と等しい――これが，この等式の意味である．自然数においては次の等式と対応する．

$\sigma\, n = n + 1$.

後者関数は 1 を足すことと等しい．これが，この等式の意味である．文字列／自然数の対比に，Cons／後者関数，連接／加法，挿入／1 といった対比が重ねられる．

左辺は帰納的定義で登場した構成子が登場し，右辺は代数演算が登場する．この意味で，この等式は計算機科学と代数の関係を端的に示している．

つぎに，挿入と拡張の関係に移ろう．挿入と拡張は次の等式を満たす．

$f^*(\eta(a)) = f(a)$.

左辺は拡張と挿入が合成された操作であり，それが拡張前の関数と等しいことを示している．これは自然数においては指数関数の等式 $x^1 = x$ に対応する．1 乗は底と等しい．これは表現が拡張と逆操作であることを示している．

さて，文字列の集合 Σ^* が自由モノイドであることを示そう．これは，Σ^* が文字の集合 Σ を基底として持つモノイドであることを意味する．基底であるためには，基底から任意のモノイドへの関数が文字列の集合からモノイドへのモノイド準同型に拡張できるという意味である．以上を示すためには，次の 4 つの条件を示せば十分である．

1. 文字列の集合 Σ^* はモノイドである――空列が単位元，連接が結合的．
2. 拡張 f^* がモノイド準同型である――単位元，乗法の構造を保存する．
3. 拡張を表現すれば拡張前に戻る．
4. 拡張は一意的である．

詳しい証明は文献[1]に譲り，ここでは補足を与えておく．以上の証明には帰納法が使用される．

文字列の集合は帰納的に定義された．そのため，文字列についての性質 P が任意の文字列で成り立つことを示すためには次の 2 つを示せば十分である．

1. ε が P を満たす．
2. α が P を満たすとき，$\alpha : a$ も P を満たす．

また，拡張が一意的であることを示す際に始代数性が有効である．拡張とは表現によって元に戻るモノイド準同型であった．この条件を満たすモノイド準同型が一意であることは自明ではない．この条件を満たすモノイド準同型の一意性は始代数の一意性によって保証される．

以上によって文字列の集合が自由モノイドであること（文字を基底に持つモノイドであること）が示される．

次章ではこの接続をモノイドから群へ継承させることを試みる．そのためには，帰納的定義の段階でいくつかの変更を施す必要がある．

4. 符号付き文字列について

本章で符号付き文字列を導入する．簡単に言えば，引き算ができる文字列である．そのため，連接の逆演算にあたる逆連接を定義することが，最終的な目標となる．連接の逆演算であるためには，同じ符号付き文字列を逆連接すると，すべての符号付き文字が打ち消し合い，最終的に空列が残ることが期待される．これは，同じ数を引くと 0 になることに対応している．

「符号付き（signed）」とは正（positive）または負（negative）が付随していることを意味する．たとえば，整数は符号付きである．各整数について符号（正または負）が付随している．さて，文字列はどうだろう．各文字について私たちは符号を考えない．正の文字または負の文字を聞いたことがあるだろうか．符号付き文字列を構成する各文字は符号を持っている．

たとえば，次のような文字列である．

$+a - b + c$，$-d + e + f$，$-g - h - i$，….

文字列を構成する各文字に符号が付いている．これが符号付き文字列である．符号が付いているのが文字であって文字列ではないことに注意されたい．正の文字と負の文字を含む符号付き文字列について，それを正もしくは負と決定することはできない．

符号が付いただけでは十分ではない．正と負が打ち消しあうために，縮約（reduction）を導入しよう．たとえば，次のような符号付き文字列は縮約される．

$+a - b + b + c = +a + c$.

左辺に注目しよう．左辺の符号付き文字列は正の b と負の b が隣接している．このように符号が異なるが文字が等しいような符号付き文字が隣接しているとき，2 つの符号付き文字は打ち消される（cancel）．

与えられた符号付き文字列について，打ち消すことができる符号付き文字をすべて打ち消すことを，縮約と呼ぶ．打ち消す順番は一意ではない．そのため，順番によって縮約された結果が等しくなることは自明ではない．とはいえ，直感的に縮約結果が等しくなることは明らかだろう．

reduction は還元を意味する単語であるが，「約分」の原語でもある．分数についても分子と分母に共通因数があれば約分される．符号付き文字列の縮約はこれと類似している．

自然数に符号を導入することで整数が得られる．その結果，引き算が全範囲できるようになった．たとえば，4 － 7 = －3 のように．

符号付き文字列にも連接の逆の演算が導入される．それを逆連接と名づけておこう．逆連接の演算子を/で表す．次のような演算として定義される．

$+a + b + c \;/\; -d + e - f = +a + b + c + f - e + d$.

2 つの符号付き文字列―― $+a+b+c$ と $-d+e-f$ ――を逆連接している．その結果は，符号付き文字列 $-d+e-f$ の順序と符号を反転させた符号付き文字列―― $+f-e+d$ ――を連接したものになった．

引き算について等式 $x - y = x + (-y)$ が成り立つことを思い出そう．引き算は足し算で置き換えられる．ここでも同様に逆連接を連接で置き換えている．

この定義は人工的であるが恣意的ではない．次が成り立つことを確認しておこう．

$a+b+c \,/\, +a+b+c = +a+b+c-c-b-a = \varepsilon$.

4.1 符号付き文字列の始代数性

さて，符号付き文字列を帰納的に定義しよう．3 節で行った文字列の定義と対応させると理解しやすい．文字列の帰納的な定義では，Cons と呼ばれる構成子が登場した．文字列

に文字を追加する機能を持っている．符号付き文字列では正の Cons と負の Cons の二つが必要となる．正の Cons は正の文字を，負の Cons は負の文字を追加する．加えて，正の Cons と負の Cons は追加するとき，最後尾の符号付き文字を可能ならば打ち消す（縮約）．

文字の集合を Σ，符号付き文字列の集合を $\Sigma^{\pm 1}$ として，次のように定義する．

1. ε は $\Sigma^{\pm 1}$ の要素である――ε は空列と呼ばれる．
2. α は $\Sigma^{\pm 1}$ の要素，a は Σ の要素であるとき，$\alpha{:}{+}a$ と $\alpha{:}{-}a$ は $\Sigma^{\pm 1}$ の要素である．
3. 任意の $\Sigma^{\pm 1}$ の要素 α と Σ の要素 a について，$\alpha{:}{+}a{:}{-}a = \alpha = \alpha{:}{-}a{:}{+}a$ が成り立つ．
4. $\Sigma^{\pm 1}$ の要素は次のようなものに限る．

文字を追加する構成子として:+と:−が登場した．これが正の Cons と負の Cons である．そして，正の Cons と負の Cons が同じ文字について繰り返されるとき，それらは打ち消し合うという条件が加えられた．

以上の帰納的定義によって，符号付き文字列は構成される．これを始代数の表現で置き換えよう．

任意の集合 X，関数 $h: X \times \Sigma \to X$，$h^{-1}: X \times \Sigma \to X$，$X$ の元 c に対して，$f(\varepsilon) = c$，$f(\alpha{:}{+}a) = h(f(\alpha), a)$，$f(\alpha{:}{-}a) = h^{-1}(f(\alpha), a)$ を満たす関数 $f^{\pm 1}: \Sigma^{\pm 1} \to X$ が一意に定まる．

ただし，h と h^{-1} は次の等式を満たすとする．

$h^{-1}(h(x, a), a)) = x = h(h^{-1}(x, a), a))$.

この条件は正の Cons と負の Cons が満たしている条件である．文献[1]ではこれを逆演算として定義した．

符号付き文字列の場合は，3つの等式を与えることで，符号付き文字列全体の集合を始域とする関数が帰納的に定義される．

例 4.1.1.（関数 length）

文字列については，その長さ（自然数）を返す関数 $length: \Sigma^{*} \to \boldsymbol{N}$ が帰納的に定義できる．符号付き文字列の長さはどうなるだろう．文字列上の関数 $length$ を参考に，$length: \Sigma^{\pm 1} \to \boldsymbol{Z}$ を次のように定義しよう．

$length(\varepsilon) = 0$, $length(\alpha{:}{+}a) = length(\alpha) + 1$, $length(\alpha{:}{-}a) = length(\alpha) - 1$.

関数の終域が整数全体の集合 $\boldsymbol{Z}$ であることに注意しよう．符号付き文字列の場合は長さが負になることがある．たとえば，符号付き文字列 $-a-b-c$ は長さが−3 である．

4.2 諸演算の定義

始代数によって符号付き文字列の集合を始域とする関数の定義方法を得た．これにより，連接，逆連接，そして拡張を定義しよう．

連接の定義は文字列の場合とほとんど変わらない．演算子を・として，次のように定義する．

$\alpha \cdot \varepsilon = \alpha$，$\alpha \cdot (\beta{:}{+}a) = (\alpha \cdot \beta){:}{+}a$，$\alpha \cdot (\beta{:}{-}a) = (\alpha \cdot \beta){:}{-}a$.

続いて，逆連接を定義しよう．この定義が符号付き文字列において最も重要である．演算子を/として，次の3つの等式により定義される．

$\alpha / \varepsilon = \alpha$，$\alpha / (\beta{:}{+}a) = (\alpha{:}{-}a) / \beta$，$\alpha / (\beta{:}{-}a) = (\alpha{:}{+}a) / \beta$.

正の文字は負の文字として，負の文字は正の文字として，一文字ずつ右から左へ送られている．このような移動を繰り返すと，順序が反転される．これによって逆連接は定義される．

拡張を定義しよう．任意の群 G と関数 $f: \Sigma \to G$ が与えられたとき，拡張された関数 $f^{\pm 1}: \Sigma^{\pm 1} \to G$ を次のように定義する．

$f^{\pm 1}(\varepsilon) = e$, $f(\alpha{:}{+}a) = f^{\pm 1}(\alpha) \cdot f(a)$, $f(\alpha{:}{-}a) = f^{\pm 1}(\alpha) \cdot f(a)^{-1}$.

負の Cons は逆元を掛けあわせている．

最後に挿入（insertion）も与えておこう．これは文字列のときと変わらない．$\eta: \Sigma \to \Sigma^{\pm 1}$ は文字 a に対して，$\varepsilon{:}{+}a$ を返す．

4.3 演算の性質

演算の性質についても，文字列の集合の場合とほとんど変わらない．正の Cons と負の Cons と連接・逆連接の間については次が成り立つ．

$\alpha{:}{+}a = \alpha \cdot \eta(a)$，$\alpha{:}{-}a = \alpha / \eta(a)$.

正の Cons は連接に，負の Cons は逆連接に置き換えることができる．

挿入と拡張も文字列の場合と同様に次が成り立つ．

$f^{\pm 1}(\eta(a)) = f(a)$.

4.4 符号付き文字列の自由性

符号付き文字列が自由群であることを示す．これは，符号付き文字列が文字の集合を基底とする群であることを示せば十分である．文字列の場合と同様に次のように証明する．

1. 符号付き文字列の集合 $\Sigma^{\pm 1}$ は群である――空列が単位元，連接が結合的，逆連接が除法の条件を満たす．
2. 拡張 $f^{\pm 1}$ が群準同型である――単位元，乗法，逆元の構造を保存する．
3. 拡張を表現すれば拡張前に戻る．
4. 拡張は一意的である．

詳しい証明は文献[1]に譲る．符号付き文字列が群の構造を持っていることは，逆連接が除法の条件を満たすことから示される．符号付き文字列において，逆連接は連接と双対的に定義することができた．そのため，証明においても連接と逆連接の性質は双対的に証明される．

また，符号付き文字列についても帰納法による証明が可能である．符号付き文字列についての性質 P が任意の符号付き文字列で成り立つことを示すためには次の2つを示せば十分である．

1. ε が P を満たす
2. α が P を満たすとき，$\alpha{:}{+}a$ と $\alpha{:}{-}a$ も P を満たす．

以上の道具立てから，符号付き文字列が自由群であることを示すのは難しくない．

4.5 考察（逆元による群の公理系の場合）

もし，通常の群の定義に沿って証明するのであれば，逆連接から逆元を定義する必要がある．逆元は次のように定義される．

$\alpha^{-1} = \varepsilon / \alpha$.

空列を符号付き文字列 α で割ると α の順序と符号を反転した符号付き文字列が得られる．これが α の逆元である．連接すると空列になる符号付き文字列である．

5. プログラミングへの応用

自由群は一つの代数系であり，任意に集合が与えられれば，それを基底とする自由群を生成することは数学的には可能である．しかし，計算機上に型として定義することは意識されていない．一方で自由モノイドは文字列（String）型として計算機上になじむ．文字列型は文字（Character）型を要素とするリスト（list）として定義できる．

ここで関数プログラミングの記法によって String 型をデータ型として与えてみよう．

```
String := Nil | Char : String.
```

これはリストとして任意の型に一般化できる．

```
[a] := [] | a : [a].
```

この定義はこの論文で与えた文字列の帰納的な定義と本質的に等しい．これを応用すれば，符号付き文字列に相当する自由群となる型を定義できる．私たちは，これを関数プログラミング言語 Haskell によって定義した．次のような定義を与えている

（http://www.isc.senshu-u.ac.jp/~ne230071/signedList.hs）.

```
[a] := a⁻¹:[a] | [] | a+:[a].
```

リスト型の構成子 Cons が 2 つ（正の Cons と負の Cons）与えられている．また，この型の値についての等号==を定義するとき，縮約の規則を導入する必要がある．連接，逆連接もこの論文で与えた定義をそのまま援用できる．以上の方法によって符号付き文字列を定義した．その結果を図 1 に載せる．

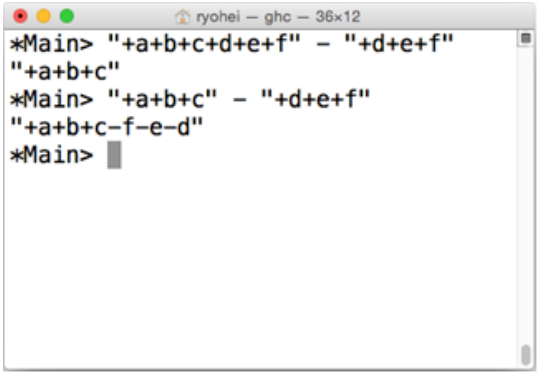

図 1　Haskell の実行例

1 行目の実行例では，*abcdef* から *def* を引いている．2 行目では，*abc* から *def* を引いている．その結果，符号と順序が反転した符号付き文字列が得られた．

6. おわりに

文字列を始代数によって定義し，それが自由モノイドであることを確認した．これにより，計算機科学（始代数）と代数（自由代数）の関係が分かり，両者の接続の方法を私たちに与えた．これを応用して，得られたのが符号付き文字列である．つまり，代数における自由群と対応関係にある，計算機科学上の帰納的構造を発見した，ということである．s

符号付き文字列はあくまでも例に過ぎず，計算機科学と代数を交差させることで，他にも生産的な結果が得られる可能性がある．

両者を架橋するために自由代数と始代数は有効である．計算機科学上で登場する帰納的な構造を持つものは始代数として定式化できる．それが代数的な構造を持っていることは珍しくない．リストであればモノイドであり，木であればマグマである．正規表現も文字から帰納的に定義される構造であり，クリーネ代数と呼ばれる代数構造を持っている．ある型が与えられ，それを格納するデータ構造が帰納的に与えられている場合は，自由代数の構造を持っている可能性がある．もし，自由代数であれば，代数の議論をデータ構造の分析に応用できるだろう．

また，代数の文脈でも改めて焦点が当てられる分野が登場するだろう．自由代数は計算機科学とは全く違う文脈で登場していた．

代数（algebra）とアルゴリズム（algorithm）は共にアラビア数学の語彙が由来である（古代ギリシアの数学などは幾何学が中心であり，代数＝計算の方法はアラビアの影響が大きい）．代数やアルゴリズムの研究とは，代数方程式を機械的に解く方法の探求を意味していた．現在の代数論は具体的な計算方法を提示するという拘束からは自由である．そのため，高度に抽象化した代数構造を対象にできる．逆に，計算機科学では具体的な計算方法を探求するため，その拘束に縛られている．同じ出発点にあった計算＝代数はこのようにして分岐してしまった．とはいえ，出発点は共通している．両者を架橋することはそれほど困難なことではないだろう．

参考文献

[1] 坂本量平 “文字列の代数”, 専修大学ネットワーク情報学部卒業論文, 2014.
http://www.isc.senshu-u.ac.jp/~ne230071/thesis.pdf

[2] Richard Bird, Oege De Moor. *Algebra of Programming*. Prentice-Hall, 1997.

[3] Saunders MacLane, Garrett Birkhoff. *Algebra*, The Macmillan Company, 1967.

[4] Richard Bird 著，山下伸夫 訳，関数プログラミング入門—Haskell で学ぶ原理と技法— ．オーム社，2012.

[5] 守屋悦朗，形式言語とオートマトン (Information Science & Engineering) ． サイエンス社，2011.

[6] Nicolas Bourbaki. *Elemtents of Mathematics, Algebra, Chapters 1-3*, Springer, 1989.

[7] Jakob Nielsen,“Die Isomorphismengruppe der freien Gruppen”, *Mathematische Annalen*, Volume 91, Issue 3-4, pp 169-209 1924.

[8] Saunders MacLane 著，三好博之，高木理 訳，圏論の基礎． 丸善出版，2012.