専修大学 ネットワーク情報学部 平成 26 年度 卒業論文

文字列の代数

ネットワーク情報学部 ネットワーク情報学科

坂本量平 (NE23-0071B)

目次

第 1 章

はじめに

コンピュータ科学と代数は文字列を研究対象として共有している．しかし，両者の視点は異なる．コンピュータ科学では，オートマトンや正規表現など，再帰的に定義される文字列を扱うため，具体的な計算方法などが明確である．このような視点を仮に「計算の視点」と呼ぼう．それに対して，代数では，自由モノイドとして文字列が登場するが，計算の視点がなく，代わりに，代数構造の分析が行われる．これを，計算の視点と対比して「代数の視点」と呼ぶとすれば，コンピュータ科学における文字列の研究には代数の視点がない．

この論文では，文字列について，計算と代数の 2 つの視点から分析するという姿勢をとる．その結果，計算の視点を維持したまま文字列を代数的に発展することができる．つまり，「符号付き文字列」の再帰的定義である．

コンピュータ科学では文字列は再帰的な定義によって与えられる．これが自由モノイドであるという代数的性質を持つことは直ちには確かめられない．逆に，代数では自由モノイドが文字列に相当することが知られている．しかし，その定義は再帰的な定義ではなく計算の視点がない．

そこで，再帰的定義によって与えられた文字列が自由モノイドであることを丁寧に確認する．代数として分析が可能になった文字列は代数的発展を示唆する．つまり，モノイドから群への拡張である．

代数の文脈では，モノイドから群への拡張は不思議なことでない．これは自由群と呼ばれる．自由群は「自由」という用語が導入されたときにすでに研究対象となっていた．これも文字列や語と呼ばれるが，コンピュータ科学で言われている文字列とは異なり，符号と呼ばれるものが付与されている．歴史的にこれはコンピュータ科学の文脈とは全く異なるところで発見されたため計算の視点はほとんどなく，あまり身近なものではない．

しかし，再帰的定義によって与えられた文字列を代数の文脈に取り込むことで，計算の視点を維持したまま，群へ発展させることが可能になる．それが「符号付き文字列」である．

この接続のために，計算的な文字列と代数的な文字列を定式化する．それぞれ，始代数と自由代数として定式化される．この定式化自体は新しくないが，両者の関係は明らかではなかった．本論では，この関係を明らかにするために，指数関数を一般化している．たとえば，文字列を指数に取る指数関数を考える．一般化した指数関数を考えることで，両者の関係は明らかになる．このような定式化は新しい試みだろう．

また，群を計算の視点から見つめなおすことで，除法の重要性が浮かび上がる．そこで，乗法と除法の関係を逆演算という概念で定式化する．新しく与えられた逆演算の定義から，群の別定義を導入する．つまり，除法による群である．群を代数の習慣通りの定義を採用せず，除法を基本とした定義を与えることで，群論を計算の視点に接続することができる．これも新しい試みである．

文字列をめぐる代数とコンピュータ科学の学問的背景は付録 A.1 で補足しているので，そちらを参照してほ

しい.

この論文の概要を説明しよう．2 章で必要となる代数の知識を準備する．それらを踏まえた上で，3 章で文字列を代数的に分析し自由モノイドであることを確認する．次に，4 章で符号付き文字列を導入する．これを文字列と同様に代数的に分析して自由群であることが確認できたら，本論の目標は達成される．

第 2 章

準備

文字列を代数的に分析すると，モノイド，始代数性，自由性という性質が浮かび上がる．そのため分析を始める前に，これらの代数的性質を準備しておく必要がある．また，その後にはモノイドから群へという移動も残されているので，群についても準備を整えよう．また，この論文では「除法による群」という通常とは異なる群の定義を導入している．通常の代数の議論に慣れている人ほど戸惑いを与えてしまうかもしれない．しかし，考えていることはとても簡単であり，符号付き文字列を考えるときにはこちらで考えるので注意してほしい．また，それに関連して「逆演算」と呼ばれる演算の定義も導入している．

2.1 モノイドと群

2.1.1 モノイド

まず，モノイドの定義を与える．

定義 2.1.1（モノイド） 集合 M がモノイドであるとは，次の条件を満たす定数 $e \in M$ と二項演算 $(\cdot) : M \times M \to M$ を持つことをいう．

結合法則 任意の M の元 x，y，z について $x \cdot (y \cdot z) = (x \cdot y) \cdot z$.
単位元 任意の M の元 x について $x \cdot e = x = e \cdot x$.

二項演算と単位元を $(\cdot)$ と e で表したが，$(+)$ と 0 という記法も使われる．$(\cdot)$ は乗法（multiplication），$(+)$ は加法（addition）と呼ばれる．通常，乗法が可換律を満たすとき加法の記法が使われる．

モノイドは代数の中でも特に単純な構造を持つものである．そのため，身近な例が多い．

例 2.1.1（自然数） 自然数の集合 $\mathbb{N}$ は足し算 $(+) : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ と 定数 0 を持つモノイドである．

例 2.1.2（文字列） Σ を文字の集合，Σ^* を Σ 上の文字からなる文字列の集合とする．このとき，Σ^* は文字列の連接をする二項演算 $(\cdot) : \Sigma^* \times \Sigma^* \to \Sigma^*$ と空語 ε を持つモノイドである．

例 2.1.3（自己準同型） 集合 X を固定する．X から X への関数（自己準同型 endomorphism）の全体を $\mathrm{End}(X)$ と表す．$\mathrm{End}(X)$ は合成 $(\circ)$ と恒等関数 id_X を持つモノイドである．

2.1.2 群

モノイドから群へ進むためには，逆元を導入する必要がある．元 x の逆元 x^{-1} とは，次のような等式を満たす元のことである．

$$x \cdot x^{-1} = e = x^{-1} \cdot x. \tag{2.1.1}$$

モノイドのすべての元が逆元を持つとき群と呼ばれる．モノイドの例として紹介した自然数，文字列，自己準同型は逆元を持たないので群ではない．群の定義は次のように与えられる．

定義 2.1.2（逆元による群） 集合 G が群であるとは，次の条件を満たす定数 e と二項演算 $(\cdot) : G \times G \to G$，単項演算 $(\_)^{-1} : G \to G$ を持つことをいう．

結合法則 任意の G の元 x，y，n について $x \cdot (y \cdot z) = (x \cdot y) \cdot z$.
単位元 任意の G の元 x について $x \cdot e = x = e \cdot x$.
逆元 任意の G の元 x について $x \cdot x^{-1} = e = x^{-1} \cdot x$.

群についてもモノイドと同様に乗法の記法と加法の記法がある．加法の記法では，二項演算を $(+)$，定数を 0，単項演算を $(-)$ で表す．

モノイドの条件に逆元の条件が加わったことに注意しよう．
代数の習慣では群の定義はこの定義（逆元による群）を指すが，この論文ではその習慣から離れて，後で異なる定義（除法による群）を導入する．

例 2.1.4（整数） 整数の集合 $\mathbb{Z}$ は足し算 $(+) : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$ と 定数 0，そして，符号を反転させる単項演算子 $(-) : \mathbb{Z} \to \mathbb{Z}$ を持つ群である．

例 2.1.5（符号付き文字列） Σ を文字の集合，$\Sigma^{\pm 1}$ を Σ 上の文字からなる符号付き文字列の集合とする．$\Sigma^{\pm 1}$ は文字列の連接をする二項演算 $(\cdot) : \Sigma^{\pm 1} \times \Sigma^{\pm 1} \to \Sigma^{\pm 1}$ と空語 ε，$(\_)^{-1} : \Sigma^{\pm 1} \to \Sigma^{\pm 1}$ を持つ群である．符号付き文字列は定義 4.1.1 で定義している．

例 2.1.6（自己同型） 集合 X を固定する．X から X への可逆な関数（自己同型 automorphism）の全体を $\mathrm{Aut}(X)$ と表す．$\mathrm{Aut}(X)$ は合成 $(\circ)$ と恒等関数 id_X を持ち，さらに，各関数 f が逆射 f^{-1} を持つので群である．ただし「可逆な関数」とは逆射を持つ関数のことである．全単射と言ってもいい．

さて，以上で，モノイドと群の定義を与えた．モノイドと群の例を比較すれば，その対応関係は明らかだろう．特に，自然数と整数の関係は分かりやすい．

表 2.1.1　モノイドと群の例

モノイド	群
自然数 $(\mathbb{N}, +, 0)$	整数 $(\mathbb{Z}, +, 0, -)$
文字列 $(\Sigma^*, \cdot, \varepsilon)$	符号付き文字列 $(\Sigma^{\pm 1}, \cdot, \varepsilon, (\_)^{-1})$
自己準同型 $(\mathrm{End}(X), \circ, \mathrm{id}_X)$	自己同型 $(\mathrm{Aut}(X), \circ, \mathrm{id}_X, (\_)^{-1})$

モノイドや群については，よく知られている代数的性質が多くあるが，本論の趣旨とは直接関係がないので

その多くは省略する．その点については，代数の入門書（文献 [4]，[9] など）を参考してほしい．

2.2 逆演算について

逆演算を導入する．たとえば，減法（引き算）は加法（足し算）の，除法（割り算）は乗法（掛け算）の逆演算である．これら二項演算について，その逆を意味するような二項演算は馴染み深いものだろう．しかし，通常の代数の議論では，逆を表す二項演算という視点はなく，逆元を表す単項演算に代わられている．たとえば，整数上の引き算 $(-) : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$ は次のように定義される．

$$m - n := m + (-n). \tag{2.2.1}$$

群 G の上で除法 $(/) : G \times G \to G$ を導入する場合にも，次のように定義されるだろう．

$$x/y := x \cdot y^{-1}. \tag{2.2.2}$$

このように，逆を表す演算は逆元によって代替可能なので代数ではあまり注目されない．しかし，ここではその習慣から離れる必要がある．符号付き文字列の定義を考えるとき，逆演算が登場する．そして，それが群であることを確かめるとき，逆元を定義するより除法を定義するほうが自然である．それが逆演算を導入する理由である．

それでは，逆演算の定義を与えよう．

定義 2.2.1（逆演算） $(\cdot)$ と $(/)$ をそれぞれ二項演算 $X \times A \to X$ とする．$(/)$ が $(\cdot)$ の逆演算であるとは，次を満たすことをいう．

任意の $x \in X$ と $a \in A$ について，

$$(x \cdot a)/a = x = (x/a) \cdot a. \tag{2.2.3}$$

この逆演算の定義は圏論の概念である積コモナドを導入することで，

$$(/) \circ (\cdot)^{\dagger} = \varepsilon_X = (\cdot) \circ (/)^{\dagger} \tag{2.2.4}$$

と表すことができる．この点については，付録 A.3 で補足しているのでそちらを参照してほしい．また，積コモナドについては文献 [3] が参考になる．

逆演算の例を見て，逆演算の条件を満たしていることを確認しよう．

例 2.2.1（引き算は足し算の逆演算） 整数上の足し算 $(+)$ と引き算 $(-)$ は次を満たす．

任意の整数 $m, n \in \mathbb{Z}$ について，

$$(m + n) - n = m = (m - n) + n.$$

例 2.2.2（割り算は掛け算の逆演算） 有理数上の掛け算 $(\times)$ と割り算 $(\div)$ は次を満たす．

任意の有理数 $m, n \in \mathbb{Q}$ について，

$$(m \times n) \div n = m = (m \times n) \div n.$$

2.2.1 除法による群

それでは，除法による群の定義を与えよう．

定義 2.2.2（除法による群） G が群であるとは，次の条件を満たす定数 e と二項演算 $(\cdot) : G \times G \to G$，二項演算 $(/) : G \times G \to G$ を持つことをいう．

乗法の結合法則 任意の G の元 x，y，z について $x \cdot (y \cdot z) = (x \cdot y) \cdot z$.
単位元 任意の G の元 x について $x \cdot e = x = e \cdot x$.
除法の結合法則 任意の G の元 x，y，z について $x \cdot (y/z) = (x \cdot y)/z$.
逆演算 任意の G の元 x，y について $(x \cdot y)/y = x = (x/y) \cdot y$.

逆元による群の定義と比較（表 2.2.1）すると「逆元」の代わりに「除法の結合法則」と「逆演算」という 2 つの条件になっている．また，1 つ目の条件「乗法の結合法則」は除法の結合法則と区別するために「乗法の」という名前に変わっているが同じ命題である．

表 2.2.1 逆元による群と除法による群

逆元による群	除法による群
乗法の結合法則	乗法の結合法則
単位元	単位元
逆元	
	除法の結合法則
	逆演算

「除法による群」と「逆元による群」が同じ代数であることを示す．次のような 2 つのステップを踏むことで，それは達成される．

1. G を定義 2.1.2（逆元による群）で与えた条件を満たす集合と仮定する．その上で，除法に相当する二項演算 $(/)$ を新しく定義し，$(/)$ が条件「除法の結合法則」,「逆演算」を満たすことを示す．
2. G を定義 2.2.2（除法による群）で与えた条件を満たす集合と仮定する．その上で，逆元を与える演算に相当する単項演算 $(\_)^{-1}$ を新しく定義し，$(\_)^{-1}$ が条件「逆元」を満たすことを示す．

ステップ 1 から行おう．まず，G を「逆元による群」であると仮定する．そのため，逆元を与える単項演算子 $(\_)^{-1} : G \to G$ を持つが，除法 $(/) : G \times G \to G$ は与えられていない．新しく二項演算子 $(/) : G \times G \to G$ を次のように定義する．

定義 2.2.3（逆元による除法）

$$x/y := x \cdot y^{-1} \tag{2.2.5}$$

右辺が「逆元による群」で与えられている演算のみで構成されていることに注意してほしい．以下，ステップ 1 が終わるまで，除法 $(/)$ はこの定義の意味で用いる．この除法 $(/)$ が「除法の結合法則」と「逆演算」の条件を満たすことを確認する．「除法の結合法則」から始めよう．

x，y，z を G の任意の元として，

$$x \cdot (y/z) = (x \cdot y)/z \tag{2.2.6}$$

が成り立つことを確かめる．除法の定義（式（2.2.5））を代入する．

$$x \cdot (y \cdot z^{-1}) = (x \cdot y) \cdot z^{-1} \tag{2.2.7}$$

これは乗法の結合法則（定義 2.1.2）から明らかである．

続いて，「逆演算」の条件を示す．同様に，x，y を G の任意の元として，

$$(x \cdot y)/y = x = (x/y) \cdot y. \tag{2.2.8}$$

が成り立つことを確かめる．除法の定義（式（2.2.5））を代入すると，

$$(x \cdot y) \cdot y^{-1} = x = (x \cdot y^{-1}) \cdot y. \tag{2.2.9}$$

となる．

これは乗法の結合法則と逆元の条件，単位元の条件（定義 2.1.2）から明らかである．

以上でステップ 1 は終了した．続いてステップ 2 を行おう．G を「除法による群」であると仮定する．その上で，逆元を与える単項演算子を次のように定義する．

定義 2.2.4（除法による逆元）

$$x^{-1} := e/x. \tag{2.2.10}$$

単位元を x で「割った」ものを x^{-1} とする，という定義である．数の語彙を借りれば，これは「逆数」に相当する．これが逆元の条件（式（2.1.1））を満たしていることを確かめよう．つまり，x を G の任意の元として，

$$x \cdot x^{-1} = e = x^{-1} \cdot x. \tag{2.2.11}$$

が成り立つことを示す．

逆元の定義（式（2.2.10））を代入する．

$$x \cdot (e/x) = e = (e/x) \cdot x. \tag{2.2.12}$$

右辺が e と等しいことは，除法の逆演算の条件（定義 2.2.2）から直ちに示される．左辺が e と等しいことは次のように示される．

$$\begin{aligned} x \cdot (e/x) &= (x \cdot e)/x \\ &= (e \cdot x)/x \\ &= e. \end{aligned}$$

1 行目は除法の結合法則から導かれる．2 行目は単位元の条件 $x \cdot e = x = e \cdot x$ から導かれている（定義 2.2.2）．

以上でステップ 2 が終了し，「逆元による群」と「除法による群」が同値であることが示された．これを定理としてまとめよう．

定理 2.2.1（逆元による群と除法による群の同値性） 定義 2.1.2 で与えた「逆元による群」と定義 2.2.2 で与えた「除法による群」は次の関係式により同じ代数をなす．

$$x/y = x \cdot y^{-1}, \tag{2.2.13}$$
$$x^{-1} = e/x. \tag{2.2.14}$$

ただし，x，y は G の任意の元とする．

この定理は，ある代数が群であることを確かめるときにどちらの公理を使っても良いことを意味している．また，どちらか一方を示せば逆元と除法が存在することが保証されている．これは符号付き文字列の分析の上で強力に機能する．

2.3 再帰と始代数

ここでは，自然数と文字列が再帰的に定義されること，それが始代数と呼ばれる概念により一般化されることを確かめよう．始代数と再帰の関係については，文献 [1] の 2 節 *Initial Algebra* に詳しく書かれている．始代数性はある条件を満たす関数の存在を保証する．そのため，演算などの構成において重要な役割を果たす．ただし，多少抽象的なので，数列という補助線を導入することで，理解を助けたい．

2.3.1 数列について

初項と漸化式を与えると，数列が一意に定まることを思い出そう．

$$a_0 = c$$
$$a_{n+1} = h(a_n).$$

c は初項，h は前の項と次の項の関係（漸化式）を表す関数である．この 2 つの条件だけで，任意の自然数 n について a_n がただ一つ存在することが保証される．これは，自然数が再帰的に定義されていることから由来している．また，c や h は，自然数上である必要はない．任意の集合 X が定数 c と関数 $h : X \to X$ を持つときでも，同様のことが言える．

これが自然数の始代数性である．定理としてまとめよう．

定理 2.3.1（自然数の始代数性） 任意の集合 X が定数 c と関数 $h : X \to X$ を持つとき，次を満たす関数 $a : \mathbb{N} \to X$ がただ一つ存在する．

$$a_0 = c \tag{2.3.1}$$
$$a_{\sigma n} = h(a_n). \tag{2.3.2}$$

σ は後者関数 (successor) と呼ばれるもので，意味は 1 を加える関数である．$(+)$ や 1 の代わりに後者関数 σ に書き換えたのは，0 や後者関数 σ の方が，$(+)$ や 1 より先に定義されるからである．

以上のように，始代数性とは，数列の語彙を使えば，初項と漸化式を与えれば数列が与えられることを意味する．これは再帰的定義の一般化になっている．

2.3.2 再帰的定義

ここでは，自然数と文字列を再帰的に定義して，それが始代数性によって定式化されることを確認する．また，自然数では帰納法による証明ができた．文字列でも同様に帰納法による証明を考えることができるので，それも紹介する．

定義 2.3.1（自然数） 自然数の集合 $\mathbb{N}$ を次のように再帰的に定義する．

- 0 は自然数である．
- n が自然数ならば，σn は自然数である．
- 自然数は，以上の操作のみによって構成される．

σ は後者関数 (successor) と呼ばれる．

定義 2.3.2（文字列） Σ を文字の集合として，Σ 上の文字からなる文字列の集合 Σ^* を次のように再帰的に定義する．

- 空語 ε は Σ^* の文字列である．
- x が Σ^* の文字列，a が Σ の文字ならば，$x : a$ は Σ^* の文字列である．
- Σ^* の文字列は，以上の操作のみによって構成される．

2 つ目の条件で使われている (:) は文字列 x の後ろに文字 a を追加する操作を表し，cons と呼ばれる（cons は Lisp を中心とした関数プログラミングの言葉で，リストと呼ばれるデータ構造における構成 (construct) から由来している）．通常は省略されて xa と表される．この記法 (:) は文献 [1] におけるリストの記法を参照している．また，ここで，「Σ の文字」とは「集合 Σ の要素」を，「Σ^* の文字列」とは「集合 Σ^* の要素」を表す．たとえば，Σ を 26 文字のローマ字の集合 $\Sigma = \{a, b, c, \cdots, z\}$ とすると，$\varepsilon : a : b : c$ は Σ^* の文字列である．$\varepsilon : a : b : c$ は abc と略記される．

以上の定義は始代数の表現で整理される．自然数は数列の語彙（初項と漸化式）を使ってすでに定式化したが，文字列と合わせるために記号を書き換えている．

定理 2.3.2（自然数の始代数性） 任意の集合 X が定数 c と関数 $h : X \to X$ を持つとき，次を満たす関数 $f : \mathbb{N} \to X$ がただ一つ存在する．

$$f(0) = c \tag{2.3.3}$$
$$f(\sigma n) = h(f(n)). \tag{2.3.4}$$

定理 2.3.3（文字列の始代数性） 任意の集合 X が定数 c と関数 $h : X \times \Sigma \to X$ を持つとき，次を満たす関数 $f : \Sigma^* \to X$ がただ一つ存在する．

$$f(\varepsilon) = c \tag{2.3.5}$$
$$f(x : a) = h(f(x), a) \tag{2.3.6}$$

以上の 2 つの定理は文献 [1] を参考にした．ただし，プログラミングの文脈で書かれているので，自然数と文字列の集合についての定理ではなく，自然数型 Nat と文字列型 $String$ における性質として書かれている．

文字列の始代数性の応用例を紹介しよう．状態空間を Q，文字集合を Σ として持つ決定性オートマトン M を考える．M は状態遷移関数 $\delta : Q \times \Sigma \to Q$ を持つ．次のように初項と漸化式を与えることで

$\delta^* : Q \times \Sigma^* \to Q$ を定義できる．

$$\delta^*(q, \varepsilon) = q$$
$$\delta^*(q, x : a) = \delta(\delta^*(x), a).$$

δ は状態 q で文字 a を受理したとき $\delta(q, a)$ に遷移することを意味する．それに対して，δ^* は状態 q で文字列 x を受理したとき $\delta^*(q, x)$ へ遷移することを意味する．

自然数で帰納法による証明ができるように，文字列についても帰納法による証明ができる．

自然数の帰納法　ある自然数上の性質 P が任意の自然数について成り立つことを，次の 2 つによって示す．

- 0 が P を満たすことを示す．
- n が P を満たすと仮定（帰納法の仮定）して，σn が P を満たすことを示す．

文字列の帰納法　ある文字列上の性質 P が任意の文字列について成り立つことを，次の 2 つによって示す．

- 空語 ε が P を満たすことを示す．
- x が P を満たすと仮定（帰納法の仮定）して，$x : a$ が P を満たすことを示す．

2.4　自由代数

文字の集合 Σ と文字列の集合 Σ^* の関係は，代数では Σ^* は Σ を基底とする自由モノイドであると整理される．ここでは，自由モノイドの定義を与えるために，再び自然数の語彙を借りる．自然数も同様に自由モノイドであり，その性質は指数関数に現れる．理解を助けるために自然数の指数関数についての性質を整理から始めて，文字と文字列の関係を自由モノイドに関する部分だけを抽出し，それを元に，自由モノイドを定義しよう．また，これと同様に自由群の形式的な定義も与える．証明については，3 章や 4 章に譲る．

2.4.1　自然数上の指数関数

定数 $x \in \mathbb{N}$ が与えられたとき，x を底とする指数関数が考えられる．定数 x と区別するために，x を底とする指数関数をプログラミングの記法を借りて次のように表そう．

$$x\hat{} : \mathbb{N} \to \mathbb{N},\ n \mapsto x^n.$$

これは自然数 n を x^n へ割り当てる関数であり，指数法則と呼ばれる次のような性質を満たす．

$$x^1 = x \tag{2.4.1}$$
$$x^0 = 1 \tag{2.4.2}$$
$$x^{m+n} = x^m \cdot x^n. \tag{2.4.3}$$

x は任意のモノイド M 上の元と考えても同様のことが言える．このとき，式 (2.4.2) は $x^0 = e$ となる．また，以上の条件を満たす関数 $x\hat{} : \mathbb{N} \to M$ はただ一つ存在である．式 (2.4.2) , (2.4.3) は，$x\hat{}$ がモノイド準同型であることを示している．モノイド準同型は次のように定義される．

定義 2.4.1（モノイド準同型）　M，M' をモノイド，f を関数 $f : M \to M'$ とする．f がモノイド準同型であるとは，次を満たすことをいう．

$$f(e) = e, \tag{2.4.4}$$
$$f(x \cdot y) = f(x) \cdot f(y). \tag{2.4.5}$$

左辺の単位元と乗法が右辺と異なることに注意しよう．左辺は M，右辺は M' の要素を表わしている．

指数関数における等式（2.4.2），(2.4.3）とモノイド準同型の条件式（2.4.4），(2.4.5）が一致していることが確認できるだろう．

群準同型もほとんど同じ形で定義されるのでここで与えておく．

定義 2.4.2（群準同型） G，G' を群，f を関数 $f : G \to G'$ とする．f が群準同型であるとは，次を満たすことをいう．

$$f(e) = e, \tag{2.4.6}$$
$$f(x \cdot y) = f(x) \cdot f(y), \tag{2.4.7}$$
$$f(x^{-1}) = f(x)^{-1}. \tag{2.4.8}$$

自然数の性質について以上のことを踏まえて再び整理しよう．これは自然数が自由モノイドであることを示している．

定理 2.4.1（自然数の自由性） 任意のモノイド M とその元 x について，次のようなモノイド準同型 $x\hat{} : \mathbb{N} \to M$ がただ一つ存在する．

$$x^1 = x. \tag{2.4.9}$$

2.4.2 文字と文字列の関係

文字と文字列の関係においても，自然数の指数関数と同じような性質がある．それを明らかにするために，新しく挿入関数（insertion）を導入しよう．Σ を文字の集合，Σ^* を文字列の集合としたとき，各文字 $a \in \Sigma$ を長さ 1 の文字列 $a \in \Sigma^*$ へ写す関数 $\eta : \Sigma \to \Sigma^*$ が存在する．これを挿入関数という．これは代数の用語である．文献 [4] では insertion，文献 [11] では「生成元の挿入」に相当する．静的型付け言語における型変換を考えると分かりやすいかもしれない．これは $Char$ から $String$ への型変換である．

$$\eta : \Sigma \to \Sigma^*,\ a \mapsto a.$$

文字列上の指数関数を導入しよう．3 章で厳密な定義を与えるので，ここではインフォーマルに定義するに留める．M を任意のモノイド，f を任意の関数 $f : \Sigma \to M,\ a \mapsto f(a)$ とする．このとき，f を底とする指数関数 $f\hat{} : \Sigma^* \to M,\ x \mapsto f^x$ を次のように定義する．

$$f^{a_1 a_2 \cdots a_n} := f(a_1) \cdot f(a_2) \cdot \cdots \cdot f(a_n).$$

ただし，$(\cdot)$ はモノイド M の乗法とする．

長さ n の文字列 $a_1 a_2 \cdots a_n$ が指数として与えられている．各 a_i は Σ 上の任意の文字である．ただし，$f^{\varepsilon} = e$ とする．

例をあげよう．$f : \Sigma \to \mathbb{N}$ を任意の文字 $a \in \Sigma$ を自然数 $1 \in \mathbb{N}$ へ割り当てる定数関数とする．

$$f(a) = 1.$$

自然数 $\mathbb{N}$ は加法 $(+)$ と定数 0 を持つモノイドである（例 2.1.1）．そのため，f を底とする指数関数 $f\hat{} : \Sigma^* \to \mathbb{N}$ を考えることができる．これは，文字列の長さを返す関数に相当する（プログラミングにおける $length$ や $size$ に相当）．例えば，長さ n の文字列 $a_1 a_2 \cdots a_n$ に対して，次のように計算される．

$$f^{a_1 a_2 \cdots a_n} = f(a_1) + f(a_2) + \cdots + f(a_n) = 1 + 1 + \cdots + 1 = n.$$

以上の定義から $f^\wedge$ がモノイド準同型であることが分かるだろう．また，任意の文字 $a \in \Sigma$ について，次が成り立つ．

$$f^{\eta(a)} = f(a). \tag{2.4.10}$$

$\eta(a)$ は文字 a からなる長さ 1 の文字列だった．これは自然数における等式 $x^1 = x$ に類似している．ここでは定数 1 と挿入関数 η が対応している．自然数と同様に，この条件を満たすモノイド準同型はただ一つしか存在しない．これは Σ^* が Σ を基底とする自由モノイドであることを意味している．

文字列の指数関数という記法はこの論文で新しく導入したものなので注意してほしい．この記法は，底 x から指数関数 $x^\wedge$ を考えるときに使われる演算子 $\wedge$ を一般化したものである．たとえば，関数 $f : A \times B \to C$ のカリー化（currying）した関数を $\tilde{f} : A \to C^B$ と表すように，関数に記号を装飾する記法を参考にしている．

2.4.3 自由モノイドと自由群

改めて自由モノイドの定義を与えよう．

定義 2.4.3（自由モノイド） モノイド Σ^* が集合 Σ を基底とする自由モノイドであるとは，ある関数 $\eta : \Sigma \to \Sigma^*$ が存在し，次を満たすことをいう．

> 任意のモノイド M と関数 $f : \Sigma \to M,\ a \mapsto f(a)$ に対して，次のようなモノイド準同型 $f^\wedge : \Sigma^* \to M$ がただひとつ存在する．
>
> $$f^{\eta(a)} = f(a). \tag{2.4.11}$$

自由群もほとんど同じである．後で導入する符号付き文字列の記号 $\Sigma^{\pm 1}$ を使って定義しよう．

定義 2.4.4（自由群） 群 $\Sigma^{\pm 1}$ が集合 Σ を基底とする自由群であるとは，ある関数 $\eta : \Sigma \to \Sigma^{\pm 1}$ が存在し，次を満たすことをいう．

> 任意の群 G と関数 $f : \Sigma \to G$ に対して，次のような群準同型 $f^\wedge : \Sigma^{\pm 1} \to G$ がただひとつ存在する．
>
> $$f^{\eta(a)} = f(a). \tag{2.4.12}$$

以上の定義は文献 [4] を参考にした．自然数の集合 $\mathbb{N}$ と整数の集合 $\mathbb{Z}$ は，それぞれ，一元集合（singleton）$1 = \{0\}$ を基底とする自由モノイドまたは自由群である（文献 [4] の Proposition1.2 と Theorem II.2）．自然数における挿入関数 $\eta : 1 \to \mathbb{N}$ は $0 \in 1$ を $1 \in \mathbb{N}$ へ返す関数，整数における挿入関数 $\eta : 1 \to \mathbb{Z}$ は $0 \in 1$ を $1 \in \mathbb{Z}$ へ返す関数である．

$$\eta : 1 \to \mathbb{N},\ \eta(0) = 1,$$
$$\eta : 1 \to \mathbb{Z},\ \eta(0) = 1.$$

自然数と整数は，1 文字からなる文字集合の文字列と符号付き文字列であると言い換えられる．以上の関係は基底と代数の 2 つの軸から次のように整理できる（表 2.4.1）．

プログラミング言語における型を考えると，次のような型と対応している（表 2.4.2）．

整数と文字列は Int と $String$ としてよく知られている．C 言語など，プログラミング言語では $String$ は $Char$ の配列（array）またはリスト（list）として実装する場合が多い．そのため，一般的には文字集合と文

表 2.4.1　自由モノイドと自由群（集合）

	モノイド	群
一元集合 1	自然数 $\mathbb{N}$	整数 $\mathbb{Z}$
文字集合 Σ	文字列 Σ^*	符号付き文字列 $\Sigma^{\pm 1}$

表 2.4.2　自由モノイドと自由群（型）

	モノイド	群
ユニット型 ()	?	Int
a	$[a]$	?

字列の関係は型 a と型 $[a]$ の関係として整理されるだろう（型 a のリストを $[a]$ と表す記法はプログラミング言語 Haskell で見られる）．自然数と符号付き文字列に対応する型は知られていない．自然数は一部のプログラミング言語で実装される．たとえば，プログラミング言語 Coq ではこれに相当する `nat` が標準で備わっている．また，コンピュータ科学（型理論など）でも自然数は典型的な研究対象である．それに対して，符号付き文字列に相当する型（符号付きリスト？）はプログラミングの文脈では登場しない．

符号付き文字列の議論はプログラミング上でそれを実装できることを示唆する．3 章と 4 章では，以上の議論を計算の視点から整理する．

第 3 章

文字列について

この章では，再帰的に定義された文字列の集合 Σ^* が自由モノイドであることを次のように証明する．まず，指数関数，連接，挿入関数といった諸演算を定義する（再帰的定義ではまだ空語 ε と cons(:) しか与えられていない）．諸演算の定義は始代数の性質によって保証される．次に，これらの諸演算の性質を整理する．これらの性質を示すためには，文字列上の帰納法が使われる．最後に，文字列が自由モノイドであることを証明する．

3.1 文字列の始代数性

文字列が再帰的に定義され，それが始代数の性質として整理できることは，再帰的定義を扱った 2.3 節で紹介したので，詳しくはそちらを参照してほしい．ここでは「文字列の再帰的定義」,「文字列の始代数性」,「文字列の帰納法」を確認する．始代数性は関数の定義を与えるために，帰納法はそれらの性質を証明するために，要請される．

定義 3.1.1（文字列） Σ を文字の集合として，Σ 上の文字からなる文字列の集合 Σ^* を次のように再帰的に定義する．

- 空語 ε は Σ^* の文字列である．
- x が Σ^* の文字列，a が Σ の文字ならば，$x:a$ は Σ^* の文字列である．
- Σ^* の文字列は，以上の操作のみによって構成される．

ただし，「Σ の文字」と「Σ^* の文字列」は，それぞれ「集合 Σ の要素」と「集合 Σ^* の要素」を表す．

定理 3.1.1（文字列の始代数性） 任意の集合 X が定数 c と関数 $h : X \times \Sigma \to X$ を持つとき，次を満たす関数 $f : \Sigma^* \to X$ がただ一つ存在する．

$$f(\varepsilon) = c \tag{3.1.1}$$
$$f(x:a) = h(f(x), a). \tag{3.1.2}$$

数列の語彙を借りれば，初項 c と漸化式 h を与えると数列が一つ定まることを意味している．

文字列の帰納法 ある文字列上の性質 P が任意の文字列について成り立つことを，次の 2 つによって示す．

- 空語 ε が P 満たすことを示す．
- x が P を満たすことを仮定（帰納法の仮定）して，$x:a$ が P 満たすことを示す．

3.2 諸演算の定義

ここでは，指数関数，連接，挿入関数の定義を与える．ただし，あくまで定義を与えているだけなので，代数的な性質を満たしているかを証明する必要がある．たとえば，ここで定義した連接が結合法則を満たすかどうかは，まだ明らかではない．それぞれの定義が正当であるのは始代数性により保証される．

また，自然数との類比から，この 3 つの諸演算は，指数関数，足し算，定数 1 に相当する．この類比は理解の助けになるだろう．

定義 3.2.1（指数関数） f を任意のモノイド M への関数 $f : \Sigma \to M,\ a \mapsto f(a)$ とする．このとき，f を底とする右指数関数 $f\hat{} : \Sigma^* \to M,\ x \mapsto f^x$ と左指数関数 $\hat{}f : \Sigma^* \to M,\ x \mapsto {}^x f$ を次のように定義する．

右指数関数

$$f^{\varepsilon} = e \tag{3.2.1}$$
$$f^{x:a} = f^x \cdot f(a). \tag{3.2.2}$$

左指数関数

$${}^{\varepsilon}f = e \tag{3.2.3}$$
$${}^{x:a}f = f(a) \cdot {}^x f. \tag{3.2.4}$$

ただし，$(\cdot)$，e はそれぞれ M の乗法と単位元とする．

通常の指数関数はここでは右指数関数と呼ばれている．それと順序が逆になっている左指数関数というのも導入している．それぞれ役割が異なる．右指数関数は自由モノイドであることを示すための指数関数に等しいが，左指数関数は連接の定義に使われる．

それぞれ，２つの等式だけで関数が定義できることは始代数性が保証している．再び，数列の語彙を借りれば，初項 c と漸化式 h に相当する部分は，それぞれ次のような定数 c と関数 $h : M \times \Sigma \to M$ である．

右指数関数

$$c := e$$
$$h(x', a) := x' \cdot f(a).$$

左指数関数

$$c := e$$
$$h(x', a) := f(a) \cdot x'.$$

次に，連接の定義を与える．そのために，cons を底とする左指数関数 $\hat{}(:)$ を考える．cons は関数 $\Sigma \to \mathrm{End}(\Sigma^*), a \mapsto (: a)$ として考えることができる．$\mathrm{End}(\Sigma^*)$ がモノイドであることは，例 2.1.3 で紹介している．$(: a) : \Sigma^* \to \Sigma^*$ は次のような関数である．

$$(: a)(x) := x : a.$$

つまり，各文字列 $x \in \Sigma^*$ に対して文字 $a \in \Sigma$ を後ろに追加した文字列 $x : a \in \Sigma^*$ を返す関数である．

cons を底とする左指数関数は次のような初項，漸化式を持つ関数である．

$$
\begin{aligned}
{}^{\varepsilon}(:) &= \mathrm{id}_{\Sigma^*} \\
{}^{x:a}(:) &= (:a) \circ {}^{x}(:).
\end{aligned}
$$

それでは，文字列上の二項演算子である連接 $(\cdot)$ を定義しよう．

定義 3.2.2（連接）

$$
x \cdot y := {}^{y}(:)(x). \tag{3.2.5}
$$

これだけでは分かりにくい．初項と漸化式がどのようになるのかを見てみよう．

$$
x \cdot \varepsilon = x \tag{3.2.6}
$$

$$
x \cdot (y:a) = (x \cdot y) : a. \tag{3.2.7}
$$

これを見ると，y について再帰的に定義されていることが分かるだろう．これは左指数関数の定義を当てはめれば，容易に確認することができる．

最後に挿入関数である．

定義 3.2.3（挿入関数） $\eta : \Sigma \to \Sigma^*$ を次のように定義する．

$$
\eta(a) := \varepsilon : a. \tag{3.2.8}
$$

挿入関数の定義は，指数関数も始代数性も使われていない．空語の後ろに追加するだけである．右辺は空語を省略して a と表されるような文字列である．

3.3 演算の性質

指数関数，連接，挿入関数の関係を整理しよう．まずは，指数関数と連接の関係から始める．これは指数法則と呼ばれるものと等しい．

定理 3.3.1（指数法則） M を任意のモノイド，f を任意の関数 $f : \Sigma \to M$ とする．このとき，f を底とする指数関数 $\Sigma^* \to M$ は任意の文字列 $x, y \in \Sigma$ について次を満たす．

右指数法則

$$
f^{x \cdot y} = f^x \cdot f^y. \tag{3.3.1}
$$

左指数法則

$$
{}^{x \cdot y}f = {}^{y}f \cdot {}^{x}f. \tag{3.3.2}
$$

証明　右指数法則（3.3.1）を示そう．これは y についての帰納法で示される．

- ε のとき
  等式は $f^{x \cdot \varepsilon} = f^x \cdot f^\varepsilon$ となる．左辺と右辺をそれぞれ展開すると同じ結果になることを示そう．
  左辺
  $$
  f^{x \cdot \varepsilon} = f^x.
  $$
  右辺
  $$
  f^x \cdot f^\varepsilon = f^x \cdot \mathrm{id}_M = f^x.
  $$
  左辺は連接の初項（3.2.7）から，右辺は指数関数の初項（3.2.2）から導かれている．

- $y : a$ のとき
  等式は $f^{x\cdot(y:a)} = f^x \cdot f^{y:a}$ となる．左辺を展開すると右辺と等しくなることを示そう．

$$\begin{aligned} f^{x\cdot(y:a)} &= f^{(x\cdot y):a} \\ &= f^{x\cdot y} \cdot f(a) \\ &= (f^x \cdot f^y) \cdot f(a) \\ &= f^x \cdot (f^y \cdot f(a)) \\ &= f^x \cdot f^{y:a}. \end{aligned}$$

  2 行目から 3 行目を導くときに，帰納法の仮定を使っている．あとは，指数関数の漸化式（3.2.2）と乗法の結合法則（定義 2.1.1）などから展開される．

左指数法則の証明は，右指数法則と同様に y についての帰納法で示されるので省略する．

定理 3.3.2（連接と挿入関数の関係） 任意の文字列 $x \in \Sigma^*$ と文字 $a \in \Sigma$ について次が成り立つ．

$$x \cdot \eta(a) = x : a. \tag{3.3.3}$$

これは，自然数における $m + 1 = \sigma m$ という等式に相当する．

証明 左辺を展開すると右辺と等しくなることを示そう．

$$x \cdot \eta(a) = x \cdot (\varepsilon : a) = (x \cdot \varepsilon) : a = x : a.$$

定理 3.3.3（指数関数と挿入関数の関係） 任意の文字 $a \in \Sigma$，モノイド M，関数 $f : \Sigma \to M$ が与えられたとき，f を底とする指数関数と挿入関数には次のような関係がある．

$$f^{\eta(a)} = f(a) = {}^{\eta(a)}f. \tag{3.3.4}$$

これは，自然数における 1 乗すると底になるという性質 $x^1 = x$ に相当する．

証明 $f^{\eta(a)}$ と ${}^{\eta(a)}f$ がそれぞれ $f(a)$ と等しくなることを示す．

$$\begin{aligned} f^{\eta(a)} &= f^{\varepsilon:a} = f^{\varepsilon} \cdot f(a) = e \cdot f(a) = f(a), \\ {}^{\eta(a)}f &= {}^{\varepsilon:a}f = f(a) \cdot {}^{\varepsilon}f = f(a) \cdot e = f(a). \end{aligned}$$

以上で，諸演算の関係の整理を終えた．簡単にまとめよう（表 3.3.1）．

表 3.3.1　諸演算の性質

	性質
連接と空語	$x \cdot \varepsilon = x$
連接と cons	$x \cdot (y : a) = (x \cdot y) : a$
連接と指数関数	$f^{x\cdot y} = f^x \cdot f^y,\ {}^{x\cdot y}f = {}^{y}f \cdot {}^{x}f$
挿入関数と cons	$x \cdot \eta(a) = x : a$
挿入関数と指数関数	$f^{\eta(a)} = f(a) = {}^{\eta(a)}f$

3.4 文字列の自由性

いよいよ文字列が自由モノイドであること，つまり，文字列の集合 Σ^* が定義 2.4.3 で与えた条件を満たすことを証明する準備を整えた．それは次のような定理として整理される．

定理 3.4.1（文字列の自由性） 文字列の集合 Σ^* は文字の集合 Σ を基底とする自由モノイドである．つまり，挿入関数 $\eta : \Sigma \to \Sigma^*$ が次を満たす．

> 任意のモノイド M と関数 $f : \Sigma \to M$ に対して，次のようなモノイド準同型 $f\hat{} : \Sigma^* \to M$ がただひとつ存在する．
> $$f^{\eta(a)} = f(a). \tag{3.4.1}$$

証明 次の 4 つの条件を示すことで十分である．

- 文字列の集合 Σ^* はモノイドである．
- 右指数関数 $f\hat{}$ はモノイド準同型である．
- 任意の文字 $a \in \Sigma$ について，$f^{\eta(a)} = f(a)$ が成り立つ．
- $f\hat{}$ はただ一つである．

まず，文字列の集合 Σ^* がモノイドであることを示そう．文字列がモノイドであるとは，連接 $(\cdot)$ と空語 ε が次の条件を満たすことである．モノイドの条件は定義 2.1.1 で与えた．

> **結合法則** 任意の Σ^* の元 x，y，z について $x \cdot (y \cdot z) = (x \cdot y) \cdot z$.
> **単位元** 任意の Σ^* の元 x について $x \cdot \varepsilon = x = \varepsilon \cdot x$.

結合法則の等式は連接の定義（定義 3.2.2）から次のように書き換えられる（両辺に文字列 x を適用することで確認できる）．

$$^{y \cdot z}(:) = {}^{z}(:) \circ {}^{y}(:).$$

この等式は左指数法則から明らかである．

次に，単位元の等式 $x \cdot \varepsilon = x = \varepsilon \cdot x$ を示す．左辺 $x \cdot \varepsilon$ が x と等しいことは，連接の定義（定義 3.2.2）から明らかである．右辺 $\varepsilon \cdot x$ が x と等しいことは，x についての帰納法で示される．

- ε のとき
  $\varepsilon \cdot \varepsilon = \varepsilon$ となる．これは明らかである．
- $x : a$ のとき
  $\varepsilon \cdot (x : a) = x : a$ を示す．左辺を展開することで確認しよう．
  $$\varepsilon \cdot (x : a) = (\varepsilon \cdot x) : a = x : a.$$
  最初の展開で帰納法の仮定を使用している．

以上で，結合法則と単位元の条件を示したので，モノイドであることが示された．次に，右指数関数 $f\hat{}$ がモノイド準同型であることを示そう．モノイド準同型の定義は定義 2.4.1 で与えた．それは任意の文字列

$x, y \in \Sigma^*$ について，次の等式を満たすことである．

$$f^{\varepsilon} = e$$
$$f^{x \cdot y} = f^x \cdot f^y.$$

これは，すでに示してある．$f^{\varepsilon} = e$ は指数関数の定義（定義 3.2.1）から，$f^{x \cdot y} = f^x \cdot f^y$ は右指数法則（定理 3.3.1）である．以上で，モノイド準同型であることが証明された．

3 つ目の条件（任意の文字 $a \in \Sigma$ について $f^{\eta(a)} = f(a)$）も，指数関数と挿入関数の関係（定理 3.3.3）としてすでに証明してある．

最後に，$f\hat{}$ がただ一つであることを示そう．$f\hat{}$ がただ一つであるとは，$f^{\eta(a)} = f(a)$ を満たすというモノイド準同型がただ一つであるという意味である．モノイド準同型 $g : \Sigma^* \to M$ が $g(\eta(a)) = f(a)$ を満たすと仮定すると，g がただ一つであることを示せばいい．その証明には始代数の性質が使われる．

g は次の等式を満たす．

$$g(\varepsilon) = e$$
$$g(x : a) = g(x) \cdot f(a).$$

$g(\varepsilon) = e$ はモノイド準同型であることから，$g(x : a) = g(x) \cdot f(a)$ は次のような式展開により確認できる．

$$g(x : a) = g(x \cdot \eta(a)) = g(x) \cdot g(\eta(a)) = g(x) \cdot f(a).$$

最初の式展開では連接と挿入関数の関係（定理 3.3.2）が使われており，次の式展開では g の仮定（$g(\eta(a)) = f(a)$）が使われている．この 2 つの等式は始代数における初項と漸化式（3.1.1），（3.1.2）に相当するものである．始代数の性質（定理 3.1.1）から，このような関数はただ一つしか存在しない．

以上で，4 つの条件をすべて示し終えたので，証明は完了した．

第 4 章

符号付き文字列について

いよいよ符号付き文字列を定義する．まずは，符号付き文字列の概観を与えたい．次のような文字の集合 Σ を考える．

$$\Sigma = \{a, b, c, \cdots, z\}.$$

このとき，Σ 上の文字からなる文字列の集合 Σ^* は次のようになる．

$$\Sigma^* = \{\varepsilon, a, b, c, \cdots, z, aa, ab, ac, \cdots, az, ba, \cdots, zz, aaa, \cdots\}.$$

それに対して，符号付き文字列の集合 $\Sigma^{\pm 1}$ は次のようになる．

$$\Sigma^{\pm 1} = \{\varepsilon, +a, -a, +b, -b, +c, -c \cdots, +z, -z, +a+a, +a-a, +a+b, \cdots\}.$$

符号付き文字列を構成する各文字に $+$ または $-$ の符号（sign）が付与されている．$+$ が付与している文字を正の文字，$-$ が付与している文字を負の文字と，そして，正または負の符号が付与している文字を符号付き文字と呼ぶ．符号付き文字列は符号付き文字からなる有限列である．

加えて符号付き文字列では縮約（reduction）の概念が含まれる．たとえば，符号付き文字列 $+a+b-b+c$ は次のように縮約される．

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

符号付き文字列 $+a-b+b-c$ では正の b と負の b が隣接している部分列がある．このように，異符号で同文字である 2 つの符号付き文字が隣接しているとき打ち消し合う．これ以上打ち消し合う符号付き文字がない符号付き文字列に変換することを縮約と呼ぶ．これは，約分（reduction）と類似している．

符号付き文字列は代数では自由群と呼ばれるものに相当している．しかし，ここでは代数の習慣に従わずにいくつか書き換えてえている．たとえば，負の a は $-a$ ではなく a^{-1} と表すことが通例である．

また，議論はまだ不十分である．たとえば，打ち消す順番を変えても同じ符号付き文字列になるかは自明ではない．詳しくは，群論についての専門書を参考にしてほしい（文献 [9] など）．

4.1　符号付き文字列の始代数性

それでは，符号付き文字列の定義を再帰的に与えよう．

定義 4.1.1（符号付き文字列） Σ を文字の集合として，Σ 上の符号付き文字からなる文字列の集合 $\Sigma^{\pm 1}$ を次のように再帰的に定義する．

- 空語 ε は $\Sigma^{\pm 1}$ の符号付き文字列である．
- x が $\Sigma^{\pm 1}$ の符号付き文字列，a が Σ の文字ならば，$x:+a$ と $x:-a$ は $\Sigma^{\pm 1}$ の符号付き文字列である．
- 任意の文字 $a \in \Sigma$ と符号付き文字列 $x \in \Sigma^{\pm 1}$ について $(x:+a):-a = x = (x:-a):+a$ が成り立つ．
- $\Sigma^{\pm 1}$ の符号付き文字列は，以上の操作のみによって構成される．

ただし，「Σ の文字」と「$\Sigma^{\pm 1}$ の符号付き文字列」は，それぞれ「集合 Σ の要素」と「集合 $\Sigma^{\pm 1}$ の要素」を表す．

文字列の定義（定義 3.1.1）と比較すると，cons が正と負の両方が用意されている．これを正の cons と負の cons と呼ぼう．また，逆演算の条件（定義 2.2.1）が 3 つ目に加えられている．これを始代数で整理すると次のようになる．

定理 4.1.1（符号付き文字列の始代数性） 任意の集合 X が定数 c と演算 $h : X \times \Sigma \to X$ とその逆演算 $h^{-1} : X \times \Sigma \to X$ を持つとき，次を満たす関数 $f : \Sigma^{\pm 1} \to X$ がただ一つ存在する．

$$f(\varepsilon) = c \tag{4.1.1}$$
$$f(x:+a) = h(f(x), a) \tag{4.1.2}$$
$$f(x:-a) = h^{-1}(f(x), a). \tag{4.1.3}$$

文字列の始代数性（定理 3.1.1）と比べると，漸化式に相当する部分が 2 つ（h と h^{-1}）に増えており，h^{-1} は h の逆演算であるという条件が加わっている．つまり，任意の $x' \in X$ と $a \in \Sigma$ について，次が成り立つ．

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

符号付き文字列上でも帰納法による証明が可能である．

符号付き文字列の帰納法 ある符号付き文字列上の性質 P が任意の符号付き文字列について成り立つことを，次の 2 つによって示す．

- 空語 ε が P を満たすことを示す．
- x が P を満たすことを仮定（帰納法の仮定）して，$x:+a$，$x:-a$ がそれぞれ P を満たすことを示す．

4.2 諸演算の定義

文字列と同様に諸演算を定義する．指数関数，連接・逆連接，挿入関数である．逆連接は連接の逆演算であり，群における除法である．整数と類比すると，指数関数，足し算，引き算，1 という対応関係にある

定義 4.2.1（指数関数） f を任意の群 G への関数 $f : \Sigma \to G,\ a \mapsto f(a)$ とする．このとき，f を底とする右指数関数 $f^{\wedge} : \Sigma^{\pm 1} \to G,\ x \mapsto f^x$ と左指数関数 ${}^{\wedge}f : \Sigma^{\pm 1} \to G,\ x \mapsto {}^x f$ を次のように定義する．

右指数関数

$$f^{\varepsilon} = e \tag{4.2.1}$$
$$f^{x:+a} = f^x \cdot f(a) \tag{4.2.2}$$
$$f^{x:-a} = f^x \cdot f(a)^{-1}. \tag{4.2.3}$$

左指数関数

$$
\begin{aligned}
{}^{\varepsilon}f &= e && (4.2.4)\\
{}^{x:+a}f &= f(a)\cdot\ {}^{x}f && (4.2.5)\\
{}^{x:-a}f &= f(a)^{-1}\cdot\ {}^{x}f. && (4.2.6)
\end{aligned}
$$

ただし，$(\cdot)$，e，$(\_)^{-1}$ はそれぞれ G の乗法，単位元と逆元とする．

これらを f を底とする右指数関数または左指数関数と呼ぶ．それでは，文字列のときと同様に初項 c と漸化式 h, h^{-1} との対応関係を整理しよう．

右指数関数

$$
\begin{aligned}
c &:= e\\
h(x', a) &:= x' \cdot f(a)\\
h^{-1}(x', a) &:= x' \cdot f(a)^{-1}.
\end{aligned}
$$

左指数関数

$$
\begin{aligned}
c &:= e\\
h(x', a) &:= f(a) \cdot x'\\
h^{-1}(x', a) &:= f(a)^{-1} \cdot x'.
\end{aligned}
$$

以上で与えた h と h^{-1} が逆演算になっていることを確かめよう．右指数関数の漸化式 h と h^{-1} で次の等式が成り立つだろうか．

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

定義から上の等式は次のようになる．

$$(x' \cdot f(a)) \cdot f(a)^{-1} = x' = (x' \cdot f(a)^{-1}) \cdot f(a).$$

この等式が成立することは，逆元の定義（定義 2.1.2）から明らかだろう．同様に左指数関数の場合も直ちに確かめられる．

続いて，連接 $(\cdot)$ と逆連接 $(/)$ の定義を与えよう．それぞれ，正の cons を底とする左指数関数 $\hat{}(: +)$ と負の cons を底とする右指数関数 $(: -)\hat{}$ として定義される．文字列上の cons が関数 $\Sigma \to \mathrm{End}(\Sigma^*)$ と考えることができたように，正の cons と負の cons は関数 $\Sigma \to \mathrm{Aut}(\Sigma^{\pm 1})$（$a \mapsto (: +a)$ と $a \mapsto (: -a)$）として考えることができる．例 2.1.6 で紹介したように $\mathrm{Aut}(\Sigma^{\pm 1})$ は群の構造を持ち，$(: +a)$ の逆元は $(: -a)$ となる．以上から，初項と漸化式は次のように整理される．

正の cons を底とする左指数関数

$$
\begin{aligned}
{}^{\varepsilon}(: +) &= \mathrm{id}_{\Sigma^{\pm 1}}\\
{}^{x:+a}(: +) &= (: +a) \circ\ {}^{x}(: +)\\
{}^{x:-a}(: -) &= (: -a) \circ\ {}^{x}(: +).
\end{aligned}
$$

負の cons を底とする右指数関数

$$
\begin{aligned}
(: -)^{\varepsilon} &= \mathrm{id}_{\Sigma^{\pm 1}}\\
(: -)^{x:+a} &= (: -)^{x} \circ (: -a)\\
(: -)^{x:-a} &= (: -)^{x} \circ (: +a).
\end{aligned}
$$

それでは，連接 $(\cdot)$ と逆連接 $(/)$ の定義を与えよう．

定義 4.2.2（連接）

$$x \cdot y := {}^{y}(:+)(x). \tag{4.2.7}$$

初項と漸化式は次のようになる．

$$\begin{aligned} x \cdot \varepsilon &= x \\ x \cdot (y : +a) &= (x \cdot y) : +a \\ x \cdot (y : -a) &= (x \cdot y) : -a. \end{aligned}$$

定義 4.2.3（逆連接）

$$x/y := (:-)^{y}(x). \tag{4.2.8}$$

初項と漸化式は次のようになる．

$$\begin{aligned} x/\varepsilon &= x \\ x/(y : +a) &= (x : -a)/y \\ y/(y : -a) &= (x : +a)/y. \end{aligned}$$

連接・逆連接の計算例を見てみよう．たとえば，次のように計算される．

$$(+a - b + c) \cdot (-c + d) = +a - b + c - c + d = +a - b + d.$$

$$(+a - b + c - d)/(+c - d) = +a - b + c - d + d - c = +a - b + c - c = +a - b.$$

挿入関数は文字列と同様に定義される．

定義 4.2.4（挿入関数） $\eta : \Sigma \to \Sigma^{\pm 1}$ を次のように定義する．

$$\eta(a) := \varepsilon : +a. \tag{4.2.9}$$

4.3 演算の性質

文字列のときと同様に，指数関数，連接・逆連接，挿入関数の関係を整理する．

定理 4.3.1（指数法則） G を任意の群，f を任意の関数 $f : \Sigma \to G$ とする．このとき，f を底とする指数関数 $\Sigma^{\pm 1} \to G$ は任意の文字列 $x, y \in \Sigma$ について次を満たす．

右指数法則

$$f^{x \cdot y} = f^{x} \cdot f^{y}, \tag{4.3.1}$$

$$f^{x/y} = f^{x} \cdot (f^{y})^{-1}. \tag{4.3.2}$$

左指数法則

$${}^{x \cdot y}f = {}^{y}f \cdot {}^{x}f, \tag{4.3.3}$$

$${}^{x/y}f = ({}^{y}f)^{-1} \cdot {}^{x}f. \tag{4.3.4}$$

証明　連接については，文字列のときと同様に証明できるので省略する．逆連接の右指数法則（4.3.2）を y についての帰納法で示す．

- ε のとき
  等式は $f^{x/\varepsilon} = f^x \cdot (f^\varepsilon)^{-1}$ となる．左辺と右辺をそれぞれ展開すると同じ結果になることを示そう．
  左辺
  $$f^{x/\varepsilon} = f^x.$$
  右辺
  $$f^x / f^\varepsilon = f^x \cdot e^{-1} = f^x \cdot e = f^x.$$
- $y : +a$ のとき
  等式は $f^{x/(y:+a)} = f^x \cdot (f^{y:+a})^{-1}$ となる．左辺を展開すると右辺と等しくなることを示そう．
  $$\begin{aligned} f^{x\cdot(y:+a)} &= f^{(x:-a)/y} \\ &= f^{x:-a} \cdot (f^y)^{-1} \\ &= (f^x \cdot (f(a))^{-1}) \cdot (f^y)^{-1} \\ &= f^x \cdot ((f(a))^{-1} \cdot (f^y)^{-1}) \\ &= f^x \cdot (f^y \cdot f(a))^{-1} \\ &= f^x \cdot (f^{y:+a})^{-1}. \end{aligned}$$
  2 行目を導くときに帰納法の仮定を使用している．また，$x^{-1} \cdot y^{-1} = (y \cdot x)^{-1}$ という性質も使用している．
- $y : -a$ のとき
  $y : +a$ と同様なので省略する．

左指数法則も右指数法則と同様に，y についての帰納法で示されるので省略する．

定理 4.3.2（連接と挿入関数の関係）　任意の文字列 $x \in \Sigma^{\pm 1}$ と文字 $a \in \Sigma$ について次が成り立つ．

$$x \cdot \eta(a) = x : +a, \tag{4.3.5}$$
$$x / \eta(a) = x : -a. \tag{4.3.6}$$

これは，整数における $m + 1 = \sigma m$ と $m - 1 = \sigma^{-1} m$ という等式に相当する．

証明　左辺を展開すると右辺と等しくなることを示そう．

$$x \cdot \eta(a) = x \cdot (\varepsilon : +a) = (x \cdot \varepsilon) : +a = x : +a,$$
$$x / \eta(a) = x / (\varepsilon : +a) = (x : -a) / \varepsilon = x : -a.$$

定理 4.3.3（指数関数と挿入関数の関係）　任意の文字 $a \in \Sigma$，モノイド G，関数 $f : \Sigma \to G$ が与えられたとき，f を底とする指数関数と挿入関数には次のような関係がある．

$$f^{\eta(a)} = f(a) = {}^{\eta(a)} f. \tag{4.3.7}$$

これは，整数における 1 乗すると底になるという性質 $x^1 = x$ に相当する．

証明　$f^{\eta(a)}$ と ${}^{\eta(a)}f$ を展開すると $f(a)$ と等しくなることを示そう．

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

以上で，諸演算の性質を整理できた．簡単にまとめよう（表 4.3.1）．

表 4.3.1　諸演算の性質

	性質
連接と空語	$x \cdot \varepsilon = x$
逆連接と空語	$x/\varepsilon = x$
連接と cons	$x \cdot (y:+a) = (x \cdot y):+a,\ x \cdot (y:-a) = (x \cdot y):-a$
逆連接と cons	$x/(y:+a) = (x:-a)/y,\ x/(y:-a) = (x:+a)/y$
連接と指数関数	$f^{x \cdot y} = f^x \cdot f^y,\ {}^{x \cdot y}f = {}^{y}f \cdot {}^{x}f$
逆連接と指数関数	$f^{x/y} = f^x \cdot (f^y)^{-1},\ {}^{x/y}f = ({}^{y}f)^{-1} \cdot {}^{x}f$
挿入関数と cons	$x \cdot \eta(a) = x:+a$
挿入関数と指数関数	$f^{\eta(a)} = f(a) = {}^{\eta(a)}f$

4.4　符号付き文字列の自由性

最後に，符号付き文字列の自由性を示す．つまり，符号付き文字列の集合 $\Sigma^{\pm 1}$ が定義 2.4.4 で与えた条件を満たすことを確かめる．

定理 4.4.1（符号付き文字列の自由性）　文字列の集合 $\Sigma^{\pm 1}$ は文字の集合 Σ を基底とする自由群である．つまり，挿入関数 $\eta : \Sigma \to \Sigma^{\pm 1}$ が次を満たす．

任意の群 G と関数 $f : \Sigma \to G$ に対して，次のような群準同型 $f\hat{} : \Sigma^{\pm 1} \to G$ がただひとつ存在する．

$$f^{\eta(a)} = f(a). \tag{4.4.1}$$

証明　次の 4 つの条件を示すことで十分である．

- 符号付き文字列の集合 $\Sigma^{\pm 1}$ は群である．
- 右指数関数 $f\hat{}$ は群準同型である．
- 任意の文字 $a \in \Sigma$ について，$f^{\eta(a)} = f(a)$ が成り立つ．
- $f\hat{}$ はただ一つである．

まず，符号付き文字列の集合 $\Sigma^{\pm 1}$ がモノイドであることを示そう．群の公理は除法による群（定義 2.2.2）を使う．文字列が群であるとは，連接 $(\cdot)$・逆連接 $(/)$ と空語 ε が次の条件を満たすことである．

乗法の結合法則　任意の符号付き文字列 $x, y, z \in \Sigma^{\pm 1}$ について $x \cdot (y \cdot z) = (x \cdot y) \cdot z$.
単位元　任意の符号付き文字列 $x \in \Sigma^{\pm 1}$ について $x \cdot \varepsilon = x = \varepsilon \cdot x$.
除法の結合法則　任意の符号付き文字列 $x, y, z \in \Sigma^{\pm 1}$ について $x \cdot (y/z) = (x \cdot y)/z$.

逆演算　任意の符号付き文字列 $x, y \in \Sigma^{\pm 1}$ について $(x \cdot y)/y = x = (x/y) \cdot y$.

乗法の結合法則と除法の結合法則は連接・逆連接の定義（定義 4.2.2 と定義 4.2.3）から次のように書き換えられる（両辺に符号付き文字列 x を適用することで確認できる）.

$$\begin{aligned} {}^{y \cdot z}(:+) &= {}^{z}(:+) \circ {}^{y}(:+), \\ {}^{y/z}(:+) &= {}^{z}(:+) \circ ({}^{y}(:+))^{-1}. \end{aligned}$$

これは左指数法則（定理 4.3.1）から明らかである．単位元の性質については，文字列のときと同様に証明できるので省略する.

逆演算を示す等式は連接・逆連接の定義から次のように書き換えられる.

$$(:-)^{y} \circ {}^{y}(:+) = \mathrm{id}_{\Sigma^{\pm 1}} = {}^{y}(:+) \circ (:-)^{y}.$$

y についての帰納法で示す.

- 空語 ε のとき
  これは明らかである.
- $y : +a$ のとき
  等式は次のようになる.

  $$(:-)^{y:+a} \circ {}^{y:+a}(:+) = \mathrm{id}_{\Sigma^{\pm 1}} = {}^{y:+a}(:+) \circ (:-)^{y:+a}.$$

  これは指数関数の定義（定義 4.2.1）から次の等式と同値である.

  $$((:-)^{y} \circ (:-a)) \circ ((:+a) \circ {}^{y}(:+)) = \mathrm{id}_{\Sigma^{\pm 1}} = ((:+a) \circ {}^{y}(:+)) \circ ((:-)^{y} \circ (:-a)).$$

  $(:+a)$ と $(:-a)$ は互いに打ち消しあうので，この等式は次のようになる.

  $$(:-)^{y} \circ {}^{y}(:+) = \mathrm{id}_{\Sigma^{\pm 1}} = {}^{y}(:+) \circ (:-)^{y}.$$

  これは，帰納法の仮定により成立する.
- $y : -a$ のとき
  上と同様に示せるので省略する.

以上で，群であることが示された．次に，右指数関数が $f\hat{}$ が群準同型であることを示そう．群準同型の定義は定義 2.4.2 で与えた．それは任意の符号付き文字列 $x, y \in \Sigma^{\pm}$ について，次の等式を満たすことである.

$$\begin{aligned} f^{\varepsilon} &= e, \\ f^{x \cdot y} &= f^{x} \cdot f^{y}, \\ f^{x^{-1}} &= (f^{x})^{-1}. \end{aligned}$$

$f^{\varepsilon} = e$ は指数関数の定義（定義 4.2.1）から，$f^{x \cdot y} = f^{x} \cdot f^{y}$ は指数法則（定理 4.3.1）から導かれる．$f^{x^{-1}} = (f^{x})^{-1}$ について示そう.

まず，左辺に現れている x^{-1} は 2.2.1 節で説明したように ε/x として定義されており，これは x の逆元になる．それを使って左辺を展開することで，等式が成り立つことが示される.

$$f^{x^{-1}} = f^{\varepsilon/x} = f^{\varepsilon} \cdot (f^{x})^{-1} = (f^{x})^{-1}.$$

最後に，$f\hat{}$ がただ一つであることを示そう．$f\hat{}$ がただ一つであるとは，$f^{\eta(a)} = f(a)$ を満たす群準同型がただ一つであるという意味である．群準同型 $g : \Sigma^* \to G$ が $g(\eta(a)) = f(a)$ を満たすと仮定すると，g がただ一つであることを示せばいい．その証明には始代数の性質が使われる．

g は次の等式を満たす．

$$\begin{aligned} g(\varepsilon) &= e, \\ g(x : +a) &= g(x) \cdot f(a), \\ g(x : -a) &= g(x) \cdot (f(a))^{-1}. \end{aligned}$$

$g(\varepsilon) = e$ は g が群準同型であることから導かれる．$g(x : +a) = g(x) \cdot f(a)$ と $g(x : -a) = g(x) \cdot (f(a))^{-1}$ は，挿入関数と連接の関係（定理 4.3.3）から次のように導かれる．

$$g(x : +a) = g(x \cdot \eta(a)) = g(x) \cdot g(\eta(a)) = g(x) \cdot f(a),$$

$$g(x : -a) = g(x/\eta(a)) = g(x) \cdot (g(\eta(a)))^{-1} = g(x) \cdot (f(a))^{-1}.$$

さて，この等式を満たす関数は始代数の性質（定理 4.1.1）からただ一つである．

以上で，4 つの条件をすべて示し終えたので，証明は完了した．

第 5 章

おわりに

本論の着想は，文字列が異なる 2 つの定義で与えられるという事実から始まっている．再帰的な定義から導かれる始代数としての文字列と基底と関係付けられたモノイドとしての文字列である．前者はコンピュータ科学の文脈で，後者は代数の文脈で登場する．この 2 つの異なる定義が同じ対象を指していたことは偶然なのか．この疑問を解決するために，両者の関係を丁寧に観察した．その結果得られたのが符号付き文字列である．

もう少し詳しく，本論の歩みを振り返ろう．まず，文字列が 2 つの異なる定義で与えられることについて，自然数における数列（初項と漸化式）と指数関数が参考になった．初項と漸化式を与えることで指数関数を定義する，という手続きを文字列へ一般化したのが，本論で行った文字列が自由モノイドであることの証明であった．

観察を終えると，モノイドから群への発展が自然に行えるようになった．モノイドで行った議論の一部を変更することによって，群への移動は容易であった．その結果，自由群に相当する符号付き文字列を再帰的に定義することに成功した．このとき群の定義も別の形への変更を要請された．逆元による群から除法による群への変更である．

再帰的定義においては逆元より除法が先行する．この除法を本論では「逆連接」と呼んでいた．連接が足し算だとすれば，逆連接は引き算である．

この議論は，自然数から整数への発展と重なっている（自然数に引き算を加えたのが整数である）．整数上の数列，帰納法はあまり馴染みがないだろう．それに対して，整数上の指数関数は不思議ではない．この整数の扱われ方の差は，そのまま，自由群についても言える．つまり，自由群という概念はコンピュータ科学の文脈で扱われないが，代数の文脈では扱われてきたということである．整数の定義も通常は再帰的に行われない．

数学の歴史において自然数から整数へ発展があった．しかし，文字列から符号付き文字列の発展は，少なくとも，コンピュータ科学上では起きていない．代数の分野で起きていたとはいえ，それは抽象的な対象でありコンピュータとの相性はよくなかった．本論では両者の関係を整理することで，コンピュータ科学上で文字列から符号付き文字列の発展を可能にした．

以上のように振り返ると，本論は，自然数から整数の発展を文字列でも行ったという一文に集約されるだろう．整数が歴史的に一般に受け入れられるのには長い時間を必要とした．そのように，符号付き文字列もコンピュータ科学上で受け入れられるのは難しいかもしれない．本論はその導きとなるだろう．

付録 A

補足

A.1 文字列の扱い

3 章では，再帰的に定義される文字列が自由モノイドであることを確認したが，この証明は他の文献を参照せずに行った．これは，まえがきで述べた，文字列がコンピュータ科学と代数の共有の研究対象ではあるが，両者は異なる視点であるとことと関係している．ここではそれについて説明しておこう．

この論文では，文字列に対して新しく符号付き文字列を導入したが，これはあくまで，コンピュータ科学の文脈において「新しい」のであって，代数では事情は異なっている．代数では文字列と符号付き文字列は自由モノイドと自由群に対応するが，代数において「文字列 string」または「語 word」として呼ばれるのは自由群である．

たとえば，文献 [9] を参照してみよう．この本は大学生用の代数の入門書であるが，モノイドではなく群から出発している．そして，自由群の意味で「語」という用語を定義している．しかし，この論文で行ったような再帰的定義ではなく「直積の直和」として与えており，それが，ここで与えた自由群の条件を満たすことを証明している．

記法も異なっている．この論文では，符号付き文字列を $\Sigma^{\pm 1}$ という記法で導入した（これ自体も他では通用しない）．しかし，この文献では集合 S を基底とする自由群を F_S として表現している．これは代数の習慣に従うか従わないかの違いを表わしている．

古典を引いてみよう．ブルバキ数学原論 [10] では自由モノイドが登場する（記法は $\mathrm{Mo}(X)$）．ここでも，上と同じように再帰的な定義ではなく，直積の直和のような定義が与えられている（そのようには書いてないが）．この論文における自由モノイドの定義に相当する命題が証明されているが，この論文とは異なる方法が採られている．

他方，コンピュータ科学（特にオートマトンやチューリングマシンを扱う形式言語理論）では，自由モノイドに相当する文字列は登場するが，自由群に相当する符号付き文字列は登場しない．たとえば，形式言語理論の入門書である文献 [8] では，文字列がモノイドであることは確認されるが，その自由性は触れられていない．

コンピュータ科学の研究対象を代数の手法を使って分析するという仕事はある．たとえば，上の入門書では「クリーネ代数」が重要な位置を占めたり，文献 [6] では普遍代数によるプログラミング意味論について書かれている．また，始代数について参照した文献 [1] は題名 *The Algebra of Programming* から分かるようにプログラミングの代数の本であり自由モノイドに相当するものも登場するが，モノイドについては触れられていない．

おそらく，事実としては知られていても，丁寧に確認されたことはなかったのだろう．少なくとも私が調べ

た限りではそのように推察される．

A.2 自然数と整数との比較

文字列から符号付き文字列の移動は，自然数から整数への移動と対応している．それらの対応関係をここでまとめてみよう．

自然数と文字列の対応関係はこのようにまとまれられる．

自然数	文字列
0	ε
σ	$(:)$
$m+n=\sigma^n(m)$	$x\cdot y=\ {}^y(:)(x)$
$1=\sigma 0$	$\eta(a)=\varepsilon:a$
$\sigma m=m+1$	$x:a=x\cdot\eta(a)$
$m+0=m=0+m$	$x\cdot\varepsilon=x=\varepsilon\cdot x$
$(m+n)+l=m+(n+l)$	$(x\cdot y)\cdot z=x\cdot(y\cdot z).$

自然数上の指数関数は左右に区別がない．そのため，足し算の定義では通常の指数関数（右指数関数）で与えられている．続いて，整数と符号付き文字列の関係を見よう．

整数	符号付き文字列
0	ε
σ	$(:+)$
σ^{-1}	$(:-)$
$\sigma^{-1}(\sigma(m))=m=\sigma(\sigma^{-1}(m))$	$(x:+a):-a=x=(x:-a):+a$
$m+n=\sigma^n(m)$	$x\cdot y=\ {}^y(:+)(x)$
$m-n=(\sigma^{-1})^n(m)$	$x/y=\ (:-)^y(x)$
$1=\sigma 0$	$\eta(a)=\varepsilon:a$
$-m=0-m$	$x^{-1}=\varepsilon/x$
$\sigma m=m+1$	$x:+a=x\cdot\eta(a)$
$\sigma^{-1}m=m-1$	$x:-a=x/\eta(a)$
$m-n=m+(-n)$	$x/y=x\cdot y^{-1}$
$m+0=m=0+m$	$x\cdot\varepsilon=x=\varepsilon\cdot x$
$(m+n)+l=m+(n+l)$	$(x\cdot y)\cdot z=x\cdot(y\cdot z)$
$(m+n)-l=m+(n-l)$	$(x\cdot y)/z=x\cdot(y/z)$

σ は後者関数（successor）と呼ばれるのに対して，σ^{-1} は前者関数（predecessor）と呼ばれる．

A.3 逆演算について

ここでは，圏論の概念である積コモナド（余単位元と余拡張）を使って，逆演算の定式化を行う．

逆演算は定義 2.2.1 で次のように与えた．

定義 A.3.1（逆演算） $(\cdot)$ と $(/)$ をそれぞれ二項演算 $X \times A \to X$ とする．$(/)$ が $(\cdot)$ の逆演算であるとは，次を満たすことをいう．

任意の $x \in X$ と $a \in A$ について

$$(x \cdot a)/a = x = (x/a) \cdot a. \tag{A.3.1}$$

積コモナドを導入すると，この定義は積コモナドの余単位元 ε_X と余拡張 $(\_)^\dagger$ によって，次のように書き換えることができる．

$$(/) \circ (\cdot)^\dagger = \varepsilon_X = (\cdot) \circ (/)^\dagger. \tag{A.3.2}$$

逆射の定義を思い出そう．$f^{-1} : X \to X$ が $f : X \to X$ の逆射であるとは次のような等式を満たすことである．

$$f^{-1} \circ f = \mathrm{id}_X = f \circ f^{-1}.$$

f の逆射 f^{-1} は f と合成すると恒等関数 id_X になる．それに対して，$(\cdot)$ の逆演算 $(/)$ は $(\cdot)$ と合成すると余単位元 ε_X になる．ただし，$(\cdot)$ と $(/)$ は直接合成をすることはできないので，余拡張をしている．

このように，積コモナドを導入することで，逆演算と演算の関係を「合成すると単位元になる」という関係に定式化することができる．

それでは，積コモナドを導入しよう．

定義 A.3.2（積コモナド） 集合 A を固定する．各集合 X について，直積 $X \times A$ を割り当てる操作を $(\_) \times A$ と表そう．このとき，積コモナドは $(\_) \times A$ と次の余単位元 ε と余拡張 $(\_)^\dagger$ の組 $((\_) \times A, \varepsilon, (\_)^\dagger)$ により与えられる．

余単位元 ε　任意の集合 X に対して，$\varepsilon_X : X \times A \to X$ を次のように定義する．

$$\varepsilon_X(x, a) = x \tag{A.3.3}$$

これは通常，射影演算子と呼ばれる演算である．

余拡張 $(\_)^\dagger$　任意の集合 X, Y と任意の関数 $f : X \times A \to Y$ について，$f^\dagger : X \times A \to Y \times A$ を次のように定義する．

$$f^\dagger(x, a) = (f(x, a), a). \tag{A.3.4}$$

積コモナドを導入すると，通常は合成できない関数の間に合成を定義することができる．たとえば，$f : X \times A \to Y$ と $g : Y \times A \to Z$ は通常は合成できない．f の余ドメインと g のドメインが等しくないからである．しかし，f を余拡張することで g と合成可能になる．

$$g \circ f^\dagger : X \times A \to Z,\ g \circ f^\dagger(x, a) = g(f(x, a), a).$$

余単位元はこの合成について単位元になる．

定理 A.3.1（余単位元）

任意の集合 X, Y と任意の関数 $f : X \times A \to Y$ について

$$f \circ \varepsilon_X^\dagger = f = \varepsilon_Y \circ f^\dagger. \tag{A.3.5}$$

これは次の命題と同値である．

任意の集合 X, Y と任意の関数 $f : X \times A \to Y$ について

$$\varepsilon_X^\dagger = \mathrm{id}_{X\times A}, \tag{A.3.6}$$
$$\varepsilon_Y \circ f^\dagger = f. \tag{A.3.7}$$

証明　式（A.3.6）から示そう．左辺に任意の $X \times A$ の元 (x, a) を与えて (x, a) になるか確かめる（$\mathrm{id}_{X\times A}$ は恒等関数だから）．

$$\varepsilon_X^\dagger(x, a) = (\varepsilon_X(x, a), a) = (x, a).$$

式（A.3.7）も同様に，左辺に任意の $X \times A$ の元 (x, a) を与えて $f(x, a)$ になるか確かめる．

$$\varepsilon_Y \circ f^\dagger(x, a) = \varepsilon_Y(f(x, a), a) = f(x, a)$$

また，この合成は結合法則を満たす．

定理 A.3.2（余拡張の結合法則）

任意の集合 X, Y, Z, W と任意の関数 $f : X \times A \to Y$，$g : Y \times A \to Z$，$h : Z \times A \to W$ について，

$$h \circ (g \circ f^\dagger)^\dagger = (h \circ g^\dagger) \circ f^\dagger. \tag{A.3.8}$$

これは次の命題と同値である．

$$(g \circ f^\dagger)^\dagger = g^\dagger \circ f^\dagger. \tag{A.3.9}$$

証明　$(g \circ f^\dagger)^\dagger$ と $g^\dagger \circ f^\dagger$ のそれぞれに，任意の $X \times A$ の元 (x, a) を与えて，同じ結果になるかを確かめる．

$$(g \circ f^\dagger)^\dagger(x, a) = (g \circ f^\dagger(x, a), a) = (g(f(x, a), a), a),$$
$$g^\dagger \circ f^\dagger(x, a) = g^\dagger(f^\dagger(x, a)) = g^\dagger(f(x, a), a) = (g(f(x, a), a), a).$$

このように，余単位元と余拡張が与える合成は，通常の関数合成のような構造を持っている．これを使うことで逆射に相当するものも定義することができ，ここで導入したのが逆演算である．逆演算の定義を再び与えよう．

定義 A.3.3（逆演算）　$(\cdot)$ と $(/)$ をそれぞれ二項演算 $X \times A \to X$ とする．$(/)$ が $(\cdot)$ の逆演算であるとは，次を満たすことをいう．

$$(/) \circ (\cdot)^\dagger = \varepsilon_X = (\cdot) \circ (/)^\dagger. \tag{A.3.10}$$

ただし，ε と $(\_)^\dagger$ は集合 A が与える積コモナドの余単位元と余拡張である．

この式 (A.3.10) が定義 A.3.1 の式 (A.3.1) と等しいことを確認しよう．3 つの関数 $(/) \circ (\cdot)^\dagger, \varepsilon_X, (\cdot) \circ (/)^\dagger : X \times A \to X$ が等しいことから，任意の $(x, a) \in X \times A$ について，$(/) \circ (\cdot)^\dagger(x, a) = \varepsilon_X(x, a) = (\cdot) \circ (/)^\dagger(x, a)$ が成り立つ．

各辺について，展開すると，

$$(/) \circ (\cdot)^\dagger(x, a) = (/)(x \cdot a, a) = (x \cdot a)/a,$$
$$\varepsilon_X(x, a) = x,$$
$$(\cdot) \circ (/)^\dagger(x, a) = (\cdot)(x/a, a) = (x/a) \cdot a.$$

よって，式（A.3.10）は，

$$(x \cdot a)/a = x = (x/a) \cdot a$$

と等しい．これは（A.3.1）と同じである．

コモナドは余単位元と余拡張を持ち，次のような条件をみたすことが要請される．

- $\varepsilon^{\dagger}_{X} = \mathrm{id}_{X \times A}$.
- $\varepsilon_{Y} \circ f^{\dagger} = f$.
- $(g \circ f^{\dagger})^{\dagger} = g^{\dagger} \circ f^{\dagger}$.

これは，余単位元と余拡張の結合法則として与えた性質と等しい．

積コモナドについては，文献 [3] が詳しい．モナド・コモナドは圏論の概念ではあるが，計算機科学，特に関数プログラミングの文脈に接続され，多少異なる進化をとげている．たとえば，以上の定義は圏論では余クライスリトリプルと呼ばれるものに等しい．モナド・コモナドは関数の型を拡張（または余拡張）することで合成を可能にする．そのため計算効果を含むような関数について，通常の関数のような合成を与えることができる．詳しくは圏論や関数プログラミングの専門書を参照されたい．

A.4 再帰的データ型と始代数

再帰的定義と始代数については，文献 [1] で詳しく書かれている．ただし，この文献は再帰的データ型の議論として書かれており，この論文はそれをコンピュータ科学の記法に大きく書き換えた．プログラミングに親しみのある人は，再帰的データ型による記法のほうが理解しやすいかもしれない．そこで，ここでは再帰的データ型の記法で始代数の議論を改めて行う．

2.3 節では自然数と文字列を集合として定義した．しかし，文献 [1] では，自然数を Nat 型，文字列を $String$ 型として定義している．

まず，Nat 型の定義から見てみよう．

定義 A.4.1（Nat 型）

$$Nat \ ::= \ zero \mid succ \ Nat.$$

これは次のような意味である．

- $zero$ は Nat 型である．
- n が Nat 型ならば，$succ \ n$ は Nat 型である．

2.3 節における自然数の定義と比較しよう．

定義 A.4.2（自然数） 自然数の集合 $\mathbb{N}$ を次のように再帰的に定義する．

- 0 は自然数である．
- n が自然数ならば，σn は自然数である．
- 自然数は，以上の操作のみによって構成される．

対応関係は明らかだろう．以上のような型定義によって，任意の型 A について，定数 $c :: A$ と関数

$h :: A \to A$ が与えられたとき，次のような関数 $f :: Nat \to A$ が定義できる．

$$\begin{aligned} f(zero) &= c \\ f(succ\ n) &= h(f\ n). \end{aligned}$$

この f は畳込演算子（fold operator）を使って $f = foldn(h, c)$ と書かれる．
次に，$String$ 型の定義を見てみよう．

定義 A.4.3（$String$ 型）

$$String\ ::=\ nil \mid cons\ (Char, String).$$

これは次のような意味である．

- nil は $String$ 型である．
- a が $Char$ 型，x が $String$ 型ならば，$cons\ (a, x)$ は $String$ 型である．

2.3 節における文字列の定義と比較しよう．

定義 A.4.4（文字列） Σ を文字の集合として，Σ 上の文字からなる文字列の集合 Σ^* を次のように再帰的に定義する．

- 空語 ε は Σ^* の文字列である．
- x が Σ^* の文字列，a が Σ の文字ならば，$x : a$ は Σ^* の文字列である．
- Σ^* の文字列は，以上の操作のみによって構成される．

Nat と同様に，任意の型 A について，定数 $c :: A$ と $h :: (Char, A) \to A$ が与えられたとき，次のような関数 $f :: String \to A$ が定義できる．

$$\begin{aligned} f(nil) &= c \\ f(cons(a, x)) &= h(a, f\ x). \end{aligned}$$

この f は畳込演算子を使って $f = foldr(h, c)$ と書かれる．

A.5 指数関数と単項演算

符号付き文字列 $\Sigma^{\pm 1}$ は群構造を持つ．そのため，任意の符号付き文字列 $x \in \Sigma^{\pm 1}$ は逆元 x^{-1} を持つ．逆元 x^{-1} は次のように計算できる．

$$x^{-1} = \varepsilon / x = (: -)^x(\varepsilon).$$

このように単項演算 $(\_)^{-1}$ は負の cons を底とする右指数関数と空語から定義される．cons の正負と指数関数の左右を考えれば $(\_)^{-1}$ を含めて，4 つの単項演算が定義できる．

それぞれの意味を考えよう．表 A.5.1 の右下（負の cons を底とする右指数関数）は x^{-1} だった．これは符号付き文字列 $x \in \Sigma^{\pm 1}$ の順序を逆にして，各符号付き文字の符号を反転した符号付き文字列を割り当てる演算である．たとえば，符号付き文字列 $(+a - b + c)^{-1}$ は次のように計算される．

$$(+a - b + c)^{-1} = -c + b - a.$$

表 A.4.1　自然数と Nat 型

自然数	Nat 型
$\mathbb{N}$	Nat
0	$zero$
σ	$succ$

表 A.4.2　文字列と $String$ 型

文字列	$String$ 型
Σ^*	$String$
Σ	$Char$
nil	ε
$(:)$	$cons$

表 A.5.1　指数関数と単項演算

	左	右
正	$x \mapsto {}^{x}(:+)(\varepsilon)$	$x \mapsto (:+)^{x}(\varepsilon)$
負	$x \mapsto {}^{x}(:-)(\varepsilon)$	$x \mapsto (:-)^{x}(\varepsilon)$

左上（正の cons を底とする左指数関数）は連接の定義から次のように計算される．

$$ {}^{x}(:+)(\varepsilon) = \varepsilon \cdot x = x. $$

つまり，恒等関数 $\mathrm{id}_{\Sigma^{\pm 1}}$ である．

右上（正の cons を底とする右指数関数）は順序を逆にする演算である．たとえば，符号付き文字列 $+a-b+c$ はこの演算によって $+c-b+a$ に割り当てられる．この演算を文字列上の演算と考えれば，コンピュータ科学において反転（reverse）と呼ばれるものに相当する．また，x^R という記法で表される（文献 [8]）．

左下（負の cons を底とする左指数関数）は各符号付き文字の符号を反転する演算である．たとえば，符号付き文字列 $+a-b+c$ はこの演算によって $-a+b-c$ に割り当てられる．この演算に特に名前はついていない（文献 [9]）．

プログラミング言語の分野では，整数の符号を反転する演算子 $(-) : x \mapsto -x$ を単項マイナス演算子（unary negation operator）や否定（negation）と呼ばれる．この記法を借りて $-x$ と表す．

以上の記法を使って整理しよう．

表 A.5.2　指数関数と単項演算の意味

	左	右
正	恒等関数 $\mathrm{id}_{\Sigma^{\pm 1}}$	順序反転 $(\_)^R$
負	符号反転 $-(\_)$	順序・符号反転 $(\_)^{-1}$

符号付き文字列 $\Sigma^{\pm 1}$ では cons に正負があったので以上の 4 つの単項演算が定義できたが，文字列 Σ^* では cons に正負がないので次の 2 つの単項演算のみ定義できる．

表 A.5.3　指数関数と単項演算（文字列）

左	右
恒等関数 id_{Σ^*}	順序反転 $(\_)^R$

A.6　圏論

始代数と自由代数の定義は圏論の概念を使って定義される．しかし，圏論の概念はあまり一般的ではないので，本論では圏論の用語を使わないで書き下した．ここでは，圏論の基本的な概念を前提として，始代数と自由代数の定義を改めて行う．

前提とする圏論の概念は，圏，対象，始対象，射，関手，随伴である．文献 [1]，文献 [11] を参考にしてほしい．

まずは，自由代数を関手の視点から整理しよう．集合の圏を **Set**，モノイドの圏を **Mon** とする．

F を集合の圏からモノイドの圏への自由関手（free functor）$F : \mathbf{Set} \to \mathbf{Mon}$，$U$ をモノイドの圏から集合の圏への忘却関手（forgetful functor）$U : \mathbf{Mon} \to \mathbf{Set}$ とする．忘却関手 U は，各モノイド M を M の台集合（underlying set）$|M|$ へ割り当てる．自由関手 F は，各集合 Σ を Σ を基底とする自由モノイド Σ^* へ割り当てる．

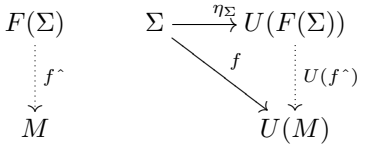
F は U の左随伴関手，または U は F の右随伴関手である．そのため，次のような射についての自然な同型が定まる．

$$\frac{f : \Sigma \to U(M) \quad \text{in } \mathbf{Set}}{f\hat{} : F(\Sigma) \to M \quad \text{in } \mathbf{Mon}}$$

これは本論における関数 $f : \Sigma \to M$ と f を底とする指数関数 $f\hat{} : \Sigma^* \to M$ の関係に相当する．圏論では，モノイド M と M の台集合 $|M|$ を区別する．そのため，本来関数 f の余ドメインである M はモノイド M の台集合であるため，$f : \Sigma \to |M|$ と表記するほうが正確である．

また，同様に本論ではモノイドとしての Σ^* と集合としての Σ^* を表記の上で区別しなかった．圏論ではこの 2 つは $F(\Sigma)$ と $U(F(\Sigma))$ として区別される．

また，F は U の右随伴なので，単位元と呼ばれる自然変換 $\eta : \mathrm{id}_{\mathbf{Set}} \to U \circ F$ を持つ．これは各集合 Σ に対して関数 $\eta_\Sigma : \Sigma \to U(F(\Sigma))$ を割り当てる．本論では単位元は挿入関数 $\eta : \Sigma \to \Sigma^*$ として登場した．単位元は次のような普遍性を持つ．つまり，任意の関数 $f : \Sigma \to U(M)$ について，あるモノイド準同型 $f\hat{} : F(\Sigma) \to M$ がただ一つ存在し，図式



を可換にする．

この図式において可換であるとは，

$$\eta_\Sigma \circ U(f\hat{}) = f \tag{A.6.1}$$

が成り立つことをいう．本論では，準同型 $f\hat{} : F(\Sigma) \to M$ と関数 $U(f\hat{}) : U(F(\Sigma)) \to U(M)$ を区別していない．これは，集合の圏の射かモノイドの圏か射かの違いである．

この普遍性は本論では挿入関数の性質として式（2.4.10）で紹介した．

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

この等式は関数の等式に書き換えると，

$$\eta \circ f\hat{} = f$$

となる．これを圏論の表記に書き換えた等式が式（A.6.1）である．

圏論と本論で使用した記法は異なっている．その対応関係を表 A.6.1 にまとめた．

表 A.6.1　記法の対応関係

	圏論の記法	本論で使用した記法
モノイド M の台集合	$U(M)$	M
文字集合 Σ を基底とする自由モノイド	$F(\Sigma)$	Σ^*
文字集合 Σ とする文字列の集合	$U(F(\Sigma))$	Σ^*
挿入関数	$\eta_\Sigma : \Sigma \to U(F(\Sigma))$	$\eta : \Sigma \to \Sigma^*$
指数関数	$f\hat{} : F(\Sigma) \to M$	$f\hat{} : \Sigma^* \to M$

文字列の自由代数性を圏論的に整理した．自由代数は自由関手と忘却関手の随伴性によって与えられる．文字列は自由モノイドであるため，随伴性は集合の圏とモノイドの圏の間にある．符号付き文字列の場合は，モノイドの圏が群の圏に置き換わる．群の圏を **Grp** とすると，忘却関手，自由関手，挿入関数，指数関数は，

$$\begin{aligned} U &: \mathbf{Grp} \to \mathbf{Set} \\ F &: \mathbf{Set} \to \mathbf{Grp} \\ \eta_\Sigma &: \Sigma \to U(F(\Sigma)) \\ f\hat{} &: F(\Sigma) \to G \end{aligned}$$

となる．

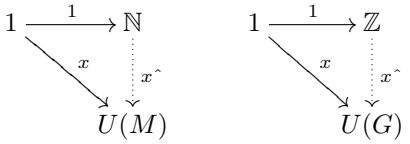
本論では符号付き文字列を $\Sigma^{\pm 1}$ と表していたが，圏論の記法では $F(\Sigma)$ と表される．記法の対応関係をまとめよう（表 A.6.2）．

表 A.6.2　記法の対応関係

	圏論の記法	本論で使用した記法
群 G の台集合	$U(G)$	G
文字集合 Σ を基底とする自由群	$F(\Sigma)$	$\Sigma^{\pm 1}$
文字集合 Σ とする符号付き文字列の集合	$U(F(\Sigma))$	$\Sigma^{\pm 1}$
挿入関数	$\eta_\Sigma : \Sigma \to U(F(\Sigma))$	$\eta : \Sigma \to \Sigma^{\pm 1}$
指数関数	$f\hat{} : F(\Sigma) \to G$	$f\hat{} : \Sigma^{\pm 1} \to G$

また，1 を要素を一つしか持たない集合（一元集合 singleton）とする．このとき，1 を基底とする自由モノイドと自由群は，それぞれ自然数と整数になる．つまり，$U \circ F(1) = \mathbb{N}$ と $U \circ F(1) = \mathbb{Z}$ が成り立つ．ただ

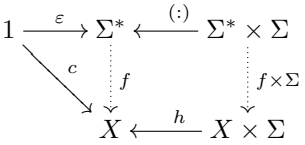
し，F はそれぞれ，自由関手 $F : \mathbf{Set} \to \mathbf{Mon}$ と $F : \mathbf{Set} \to \mathbf{Grp}$ を表す．図式で表せば，



となる．

次に始代数について整理する．文字集合 Σ を文字列の集合 Σ^* へ割り当てる操作は関手 $(\_)^* : \mathbf{Set} \to \mathbf{Set}$ と考えられる．この関手は次のような始代数によって定義される．

集合 Σ を固定する．このとき，ある集合 Σ^*，関数 $\varepsilon : 1 \to \Sigma^*$ と $(:) : \Sigma^* \times \Sigma \to \Sigma^*$ が次のような普遍性を持つ．つまり，任意の集合 X，関数 $c : 1 \to X$ と $h : X \times \Sigma \to X$ が与えられたとき，ある関数 $f : \Sigma^* \to X$ がただ一つ存在し，図式



を可換にする．この可換性を関数合成に関する等式に書き換えると，

$$f \circ \varepsilon = c,\ f \circ (:) = h \circ f \times \Sigma$$

となる．これは，文字列の始代数性として与えた初項と漸化式に関する次の 2 つの等式（3.1.1），（3.1.2）に相当する．

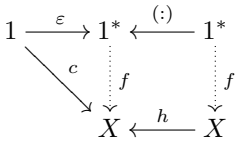
$$f(\varepsilon) = c,\ f(x : a) = h(f(x), a).$$

定数 $c \in X$ が関数 $c : 1 \to X$ に置き換わっていることに注意しよう．集合 1 は要素を一つしか持たないため，関数 c は X の要素をただ一つ割り当てる関数である．よって，X 上の定数と同一視できる．

この普遍性を始代数性という．始代数とはある関手 F が誘導する F 代数の圏 $\mathbf{Alg}(F)$ の始対象を意味する．また，一意的に定まる射を catamorphism と呼ぶ．バナナ括弧と呼ばれる括弧によって $(\!|h, c|\!) : \Sigma^* \to X$ と表される（文献 [1]，[2] など）．

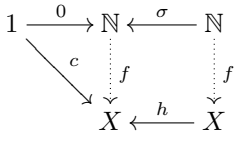
ただし，ここでは関手 F による代数の圏ではなく，図式 $1 \longrightarrow X \longleftarrow X \times \Sigma$ が誘導する圏の始対象と考える．文字列の場合は関手でも図式でも同じ圏を指すが，次の符号付き文字列の議論における始対象が属す圏は関手では表現できないからである．

Σ を 1 と置いてみよう．1 は直積について単位元であるため，1 の始代数 1^* の普遍性は図式



が可換であることによって表される．

他方，自然数の始代数性は図式



が可換であることによって表される．

対応関係は明らかだろう．この対応は $1^* = \mathbb{N}$ を意味する．

各集合 Σ に対して，以上のような始代数性を持つ集合 Σ^* を割り当てる操作は関手 $(\_)^* : \mathbf{Set} \to \mathbf{Set}$ になる．

3 章では，始代数によって定義される文字列が自由モノイドであることを確認した．圏論の表現では図式

$$\begin{array}{ccc} \mathbf{Set} & \xrightarrow{F} & \mathbf{Mon} \\ & \searrow^{(\_)^*} & \downarrow{\scriptstyle U} \\ & & \mathbf{Set} \end{array}$$

が可換であることを示したと言い換えられる．関手 $(\_)^*$ は始代数を与える関手であり，関手 $U \circ F$ は自由関手と忘却関手の合成関手である．この 2 つの関手の関係を示したのは本論の一つの成果だろう．もう一つの成果は以上の関係を群まで拡張したことである．

4 章で行ったモノイドから群への移動は，圏論の表現では図式

$$\begin{array}{ccc} \mathbf{Set} & \xrightarrow{F} & \mathbf{Grp} \\ & \searrow^{(\_)^{\pm 1}} & \downarrow{\scriptstyle U} \\ & & \mathbf{Set} \end{array}$$

を可換にする関手 $(\_)^{\pm 1}$ を与えたと言い換えられる．本論では関手 $(\_)^{\pm 1} : \mathbf{Set} \to \mathbf{Set}$ は次のような始代数として与えている．

集合 Σ を固定する．このとき，ある集合 $\Sigma^{\pm 1}$，関数 $\varepsilon : 1 \to \Sigma^{\pm 1}$，$(: +) : \Sigma^{\pm 1} \times \Sigma \to \Sigma^{\pm 1}$ と $(: -) : \Sigma^{\pm 1} \times \Sigma \to \Sigma^{\pm 1}$ が次のような普遍性を持つ．つまり，任意の集合 X，関数 $c : 1 \to X$，$h : X \times \Sigma \to X$ と $h^{-1} : X \times \Sigma \to X$ が与えられたとき，ある関数 $f : \Sigma^{\pm 1} \to X$ がただ一つ存在し，図式

$$\begin{array}{ccccc} 1 & \xrightarrow{\varepsilon} & \Sigma^{\pm 1} & \underset{(:-)}{\overset{(:+)}{\leftleftarrows}} & \Sigma^{\pm 1} \times \Sigma \\ & \searrow^{c} & \downarrow{\scriptstyle f} & & \downarrow{\scriptstyle f \times \Sigma} \\ & & X & \underset{h^{-1}}{\overset{h}{\leftleftarrows}} & X \times \Sigma \end{array}$$

を可換にする．ただし，$(: -)$ と h^{-1} は，それぞれ $(: +)$ と h の逆演算である．この可換性を関数合成に関する等式に書き換えると，

$$f \circ \varepsilon = c,\ f \circ (: +) = h \circ f \times \Sigma,\ f \circ (: -) = h^{-1} \circ f \times \Sigma$$

となる．これは，符号付き文字列の始代数性として与えた初項と漸化式に関する次の 3 つの等式（4.1.1），(4.1.2），(4.1.3）に相当する．

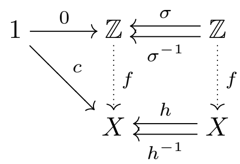
$$f(\varepsilon) = c,\ f(x : +a) = h(f(x), a),\ f(x : -a) = h^{-1}(f(x), a).$$

再び，Σ を 1 と置いてみよう．このとき，逆演算は逆射と同一視される．$1^{\pm 1}$ についての図式は，

$$\begin{array}{ccccc} 1 & \xrightarrow{\varepsilon} & 1^{\pm 1} & \underset{(:-)}{\overset{(:+)}{\leftleftarrows}} & 1^{\pm 1} \\ & \searrow^{c} & \downarrow{\scriptstyle f} & & \downarrow{\scriptstyle f} \\ & & X & \underset{h^{-1}}{\overset{h}{\leftleftarrows}} & X \end{array}$$

となる．

これは整数についての図式



に対応する．

以上の始代数は図式 $1 \longrightarrow X \Longleftarrow X \times \Sigma$（ただし，2 つの射 $X \times \Sigma \to X$ は逆演算の関係にある）が誘導する圏の始対象である．

$(\_)^*$ と $(\_)^{\pm 1}$ は関手である．そのため，集合だけではなく関数に対しても割り当てを与えなければならない．つまり，任意の関数 $f : \Sigma_1 \to \Sigma_2$ に対して，関数 $f^* : \Sigma_1^* \to \Sigma_2^*$，$f^{\pm 1} : \Sigma_1^{\pm 1} \to \Sigma_2^{\pm 1}$ を定義する必要がある．

$f^* : \Sigma_1^* \to \Sigma_2^*$ は次の初項と漸化式で与えられる．

$$f^*(\varepsilon) = \varepsilon$$
$$f^*(x : a) = f^*(x) : f(a)$$

$f^{\pm 1} : \Sigma_1^{\pm 1} \to \Sigma_2^{\pm 1}$ は次の初項と漸化式で与えられる．

$$f^{\pm 1}(\varepsilon) = \varepsilon$$
$$f^{\pm 1}(x : +a) = f^{\pm 1}(x) : +f(a)$$
$$f^{\pm 1}(x : -a) = f^{\pm 1}(x) : -f(a)$$

f^* と $f^{\pm 1}$ は，それぞれモノイド準同型，群準同型である．

A.7　文字列の長さについて

文字列には長さ（length）がある．つまり，各文字列 $x \in \Sigma^*$ について，x の長さ $length(x) \in \mathbb{N}$ が定まる．たとえば，文字列 $abc \in \Sigma^*$ の長さは 3 である．

これは，関数 $length : \Sigma^* \to \mathbb{N}$ として定式化される．この $length$ は圏論の概念により次のように定義できる．一元集合 1 は集合の圏の終対象である．つまり，任意の集合 Σ について，関数 $\Sigma \to 1$ はただ一つしか存在しない．この関数を $! : \Sigma \to 1$ と表そう．この関数を関手 $(\_)^*$ によって $!^* : \Sigma^* \to 1^*$ へ割り当てる．$1^* = \mathbb{N}$ だった（A.6 節）．よって，$!^*$ は自然数への関数 $!^* : \Sigma^* \to \mathbb{N}$ である．これが $length$ である．

それでは，符号付き文字列の場合はどうなるだろう．文字列と同様に考えると，符号付き文字列の長さを表す関数は $!^{\pm 1} : \Sigma^{\pm 1} \to \mathbb{Z}$ となる．これは符号付き文字列が負の長さを持つことを示唆している．たとえば，符号付き文字列 $+a - b + c \in \Sigma^{\pm 1}$ を考えよう．$+a - b + c$ の長さは 2 である．これは，正の文字と負の文字の個数の差を表わしている．そのため，符号付き文字列 $-a - b - c \in \Sigma^{\pm 1}$ の長さは -3 である．

A.8　単項演算と二項演算

本論では群の定義を逆元ではなく除法から行った．その理由は，符号付き文字列の諸演算の定義の順番にある．

改めて，4 章で行った諸演算の定義を振り返ろう．最初に空語 ε，正の cons(: +)，負の cons(: −) が定義された．その次に，指数関数を定義した．しかし，指数関数の定義は必須ではない．連接・逆連接を指数関数で定義したが，初項と漸化式を与えることで，指数関数を回避できる．連接と逆連接の定義をしている部分（定義 4.2.2，4.2.3）では，指数関数による定義が初項と漸化式による定義によって言い換えられることを示した．ここで逆連接が群における除法に相当することに注意しよう．この定義では逆元を割り当てる単項演算を必要としていない．

逆元を割り当てる単項演算は空語と逆連接を使って次のように定義できる．

$$x^{-1} := \varepsilon / x \tag{A.8.1}$$

このように定義できることは，2.2 節で確認した．

逆元を逆連接より先に定義することはできるだろうか．おそらくそれはできない．その根拠を数学的な証明によって示すことはできないが，経験的な事実としては示すことができる．この節ではそれを行おう．

プログラミング言語 Haskell を参照する．Haskell ではリストという型（型構成子）が存在する．これは本論における文字集合から文字列集合を構成する操作に対応している．詳しくは文献 [5] を参照されたい．

例えば，リスト上の連接 (++) は次のように定義されている．

```
(++) :: [a] -> [a] -> [a]
(++) []     ys = ys
(++) (x:xs) ys = x : xs ++ ys
```

文字列上の連接の定義 3.2.2 と同じ定義であることが確認できるだろう．

Haskell の標準関数の中でリスト上の単項演算がどのように定義されているかを見てみよう．`last`，`init`，`reverse`，`nub`，`sum`，`product` は次のように定義されている．

```
last                    :: [a] -> a
last []                 =  errorEmptyList "last"
last (x:xs)             =  last' x xs
  where last' y []     = y
        last' _ (y:ys) = last' y ys

init                    :: [a] -> [a]
init []                 =  errorEmptyList "init"
init (x:xs)             =  init' x xs
  where init' _ []     = []
        init' y (z:zs) = y : init' z zs

reverse                 :: [a] -> [a]
reverse l =  rev l []
  where
    rev []     a = a
    rev (x:xs) a = rev xs (x:a)
```

```
nub                     :: (Eq a) => [a] -> [a]
nub l                   = nub' l []
  where
    nub' [] _           = []
    nub' (x:xs) ls
        | x `elem` ls   = nub' xs ls
        | otherwise     = x : nub' xs (x:ls)

sum                     :: (Num a) => [a] -> a
sum     l       = sum' l 0
  where
    sum' []     a = a
    sum' (x:xs) a = sum' xs (a+x)

product                 :: (Num a) => [a] -> a
product l       = prod l 1
  where
    prod []     a = a
    prod (x:xs) a = prod xs (a*x)
```

last と init は空ではないリストの最後の要素，または最後の要素を除いたリストを返す演算，reverse は要素の順序を逆にしたリストを返す演算，nub はリストの要素の重複を除く演算，sum，product は総和，総積である．以上の定義は Data.List のソース（http://hackage.haskell.org/package/base-4.7.0.1/docs/src/GHC-List.html）を参考にした．定義の方法は他にもあるが，特に注目したい方法のものをここに抜粋している．

それぞれ，where によって別名の関数（last'，init'，rev，nub'，sum'，prod）を局所定義していることに注意してほしい．局所定義された別名の関数は二項演算である．このように，ある二項演算を局所定義しておいて，単項演算を定義するということがしばしばある．この現象は，本論における逆元を割り当てる演算を定義するときにも起きている．また，last，init はこれに当てはまらないが，単項演算が二項演算に単位元（空列，0，1）を適用して与えられていることも共通している．

このような現象は本論の表現で言えば「計算の視点」によって発見されたものである．代数において，二項演算と単項演算にこのような関係があり，それらを計算したいときには以上の議論が参考になるかもしれない．たとえば，本論のように，単項演算を定義するのが困難であるとき，二項演算から定義すればよい．また，新しく浮かび上がった二項演算を「代数の視点」で分析することもまた無駄ではないだろう．本論では除法の再発見があった．局所定義された二項演算は，定義が局所的であること，命名が二次的（' の付与や省略形）からも，特に重要視されていない．

このように「計算の視点」と「代数の視点」を交差，往復することで，互いに有益な示唆を与えることがある．本論の符号付き文字列や除法はその一例であった．

参考文献

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

[2] Erik Meijer, Maarten Fokkinga and Ross Paterson. Functional Programming with Bananas, Lenses, Envelopes and Barbed Wire. In *Functional Programming Languages and Computer Architecture* ,*Lecture Notes in Computer Science*, Volume 523, Pages 124-144, Springer-Verlag, 1991.

[3] Dominic Orchard. Should I use a Monad or a Comonad?, 2012. `http://www.cl.cam.ac.uk/~dao29/drafts/monad-or-comonad-orchard11-draft.pdf`.

[4] Saunders MacLane and Garrett Birkhoff. *Algebra*, 3rd edition, AMS-Chelsea, 1999.

[5] 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.

[6] 蓮尾一郎「計算機科学と代数学 プログラム意味論と普遍代数学」．2014．`http://www-mmm.is.s.u-tokyo.ac.jp/~ichiro/papers/algPLJapaneseJan2014.pdf`.

[7] 勝股審也「圏論の歩き方 (第 5 回) モナドと計算効果」．数学セミナー 2011 年 12 月号， 日本評論社.

[8] 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.

[9] 雪江明彦『代数学 1　群論入門 (代数学シリーズ) 』．日本評論社， 2010.

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

[11] Saunders Mac Lane 著，三好博之，高木理 訳『圏論の基礎』．丸善出版，2012.