１．　２章, 3 章補足定義

１．１

要素 a，b からなる集合（あるいはアルファベット）Σを $\Sigma=\{a, b\}$と書く。Σの要素を 2 個連ねることによって得られる集合を

$$\Sigma\cdot\Sigma=\Sigma^2=\{xy \mid x, y\in\Sigma\}=\{aa, ab, ba, bb\}$$

と定義する。一般に，集合 Σ, Γに対して，その積（あるいは連接）を

$$\Sigma\cdot\Gamma=\{xy \mid x\in\Sigma, y\in\Gamma\}$$

と定義する。なお，xy を x と y の連接（concatenation）という。集合 Σの要素を i 個（ただし，$i>0$）連接することによって得られる集合を

$$\Sigma^i=\begin{cases}\Sigma & (i=1\text{ のとき})\\ \Sigma\cdot\Sigma^{i-1} & (i>1\text{ のとき})\end{cases}$$

と定義する。同様に，要素 $a\in\Sigma$を i 個連接した記号列を a^i で表す。

∪と∩はそれぞれ集合和と共通部分集合をとる演算子である：

$$\Sigma\cup\Gamma=\{x \mid x\in\Sigma\text{ 又は }x\in\Gamma\}$$

$$\Sigma\cap\Gamma=\{x \mid x\in\Sigma\text{ かつ }x\in\Gamma\}$$

すべての Σ^i（ただし，$i>0$）の要素からなる集合を

$$\Sigma^+=\cup_{i=1}^{\infty}\Sigma^i=\Sigma\cup\Sigma^2\cup\Sigma^3\cup\cdots$$

と定義する。Σ^+ の要素を語または系列と呼ぶ。$x\in\Sigma^i$のとき，語 x の長さは i であり，$|x|=i$ と書く。例えば，$x=ab\in\Sigma^2$ ならば，その長さは 2 である。長さ 0 の語を空語と呼び，ε で表す。すなわち，$|\varepsilon|=0$。また，すべての $x\in\Sigma^+\cup\{\varepsilon\}$ に対して，$\varepsilon x=x=x\varepsilon$とする。すなわち，$\varepsilon$ は連接演算のもとでの単位元である。

Σの閉包（closure）またはクリーネ（Kleene）閉包を

$$\Sigma^*=\Sigma^+\cup\{\varepsilon\}$$

と定義する。空集合を ϕ で表す。ここで ϕ は$\{\varepsilon\}$と異なることに注目されたい。$\phi\cup\Sigma=\Sigma=\Sigma\cup\phi$，$\phi\cdot L=L\cdot\phi=\phi$が成立する。

１．２

集合 S 上の関係 R は $S\times S$ の部分集合である（$R\subseteq S\times S$）。$a, b\in S$ に対して，$(a, b)\in R$ のとき，aRb とも書く。関係が反射的（reflexive），推移的（transitive），対称的（symmetric），非対称的（asymmetric）とはつぎのように定義される。

R は反射的$\Leftrightarrow\forall a\in S$: aRa

R は推移的$\Leftrightarrow\forall a, b, c\in S$: aRb かつ bRc ならば，aRc

R は対称的$\Leftrightarrow\forall a, b\in S$: aRb ならば，bRa

R は非対称的$\Leftrightarrow\forall a, b\in S$: aRb ならば，bRa でない

関係 R の推移閉包 R^+をつぎのように定義する。

(1)　$aRb\Rightarrow aR^+b$

(2)　aR^+b かつ $bRc\Rightarrow aR^+c$

(3)　(1)と(2)から導けないものは R^+の元でない

R の反射的推移閉包 R^*を $R^+ \cup \{(a, a)|a \in S\}$と定義する。$R$ が同値関係であるとは，R が反射的，推移的および対称的であるときである。

１．３

【定義】集合（アルファベット）Σ上の正規集合 (または正規言語) はつぎのように定義される。

1) $\phi, \{\varepsilon\}$ は正規集合である。
2) 任意の $a \in \Sigma$ に対して，$\{a\}$ は正規集合である。
3) A と B が正規集合であるならば，$A \cup B, A \cdot B, A^*$ は正規集合である。
4) 上の 1)～3)を適用して得られるものだけが正規集合である。

正規集合は通常，つぎのように正規表現（regular expression, re）を用いて表される。

【定義】集合 Σ上の正規集合の正規表現はつぎのように与えられる。

1) 集合 ϕ と $\{\varepsilon\}$ の正規表現はそれぞれ ϕ と ε である。
2) 任意の $a \in \Sigma$ に対して，集合$\{a\}$の正規表現は a である。
3) 正規集合 A と B の正規表現をそれぞれ R_A と R_B とするとき，$A \cup B$, $A \cdot B$ および A^* の正規表現はそれぞれ$(R_A) \mid (R_B)$, $(R_A) \cdot (R_B)$および$(R_A)^*$ である。

ここで，演算子の優先順位は ＊ > ・ > | であり，不必要なかっこは除去できる。なお，連接演算子は省略できる。

【例】符号なし整数の正規表現はつぎのように与えられる。

数字=0|1|2|3|4|5|6|7|8|9
符号なし整数=数字・数字*

２．　λ算法

λ算法を定義するために，以下の準備を必要とする。

【定義】（束縛変数）

1. 変数 x は変数 y（$y=x$ でもよい）中では束縛されない。
2. 変数 x がλ式 M 又は N 中で束縛されているならば，x は MN 中で束縛されている。
3. 変数 x がλ式 M 中で束縛されているかまたは $x=y$（ただし，$y \in X$）であれば，x は $\lambda y.M$ 中で束縛されている。
4. 変数 x がλ式 M 中で束縛されているならば，λ式(M)中でもそうである。

【定義】（自由変数）

1. 変数 x は変数 x 中で自由である。

2. 変数 x がλ式 M または N 中で自由ならば，x は MN 中で自由である。
3. 変数 x がλ式 M 中で自由で，また変数 y が x と異なるならば，x は $\lambda y.M$ 中で自由である。
4. 変数 x がλ式 M 中で自由ならば，(M)中でも自由である。

例として，λ式 $P=(\lambda x.xy)x$ 中では y は自由であるが，x の二つの出現（すなわち，y の左隣とλ式の最も右側の出現）に対して，前者は束縛されており，後者は自由である。

まず，λ式 M 中に出現する自由変数 x にλ式 N を代入する方法について述べる。この代入で注意しなければいけないことは N 中の自由変数を束縛しないように，M 中の束縛変数を変更する必要がある。

【定義】 $x\in X, M, N\in E_\lambda$とする。† $[x\to N]M$ はつぎのように定義されるλ式 M' である。

(1) $M\in X$ のとき
　$x=M$ ならば　$M'=N$
　$x\neq M$ ならば　$M'=M$

(2) $M\notin X$ のとき
　(2.1) $M=M_1M_2$ のとき
　　$M'=([x\to N]M_1)([x\to N]M_2)$
　(2.2) $M=\lambda y.M_1$ のとき
　　(2.2.1) $x=y$ ならば　$M'=M$
　　(2.2.2) $x\neq y$ のとき
　　(a) M_1 中に自由な x の出現があり，かつ N 中で自由な y の出現があるならば
　　$M'=\lambda z.[x\to N]([y\to z]M_1)$
　　ただし，z は N 中にも，M_1 中にも出現しない変数である。
　　(b) (a)以外のとき
　　$M'=\lambda y.[x\to N]M_1$

λ算法の変換規則

α変換：　$\lambda x.M\in E_\lambda, y\in X$ とする。
　M 中に自由な y の出現がなければ，$\lambda x.M\to_\alpha \lambda y.[x\to y]M$

β変換：　$(\lambda x.M)N\to_\beta [x\to N]M$
　ただし，$(\lambda x.M)N\in E_\lambda$

η変換：　M 中に自由な x の出現がなければ，$\lambda x.Mx\to_\eta M$
　ただし，$\lambda x.Mx\in E_\lambda$

† X は変数の集合，E_λはλ式の集合である。

３．　７章補足

逆ポーランド（reverse Polish）記法

後置（postfix）記法ともいい，演算子を後置きして式を表現する方法。

【例】 −を単項演算子，+，∗を二項演算子，*a, b, c* を識別子（変数名）または定数とするとき，以下の通常の式（infix 記法の式）を逆ポーランド記法の式に変換（⇒）すると，次のようになる。

$$a+b \Rightarrow ab+$$
$$-a+b+c \Rightarrow a-b+c+$$
$$(a+b)*c \Rightarrow ab+c*$$
$$-(a*b) \Rightarrow ab*-$$
$$a+b*c \Rightarrow abc*+$$
$$a+b*(b+c)*c \Rightarrow abbc+*c*+$$

ここで，演算子の順位は−>∗>+ であり，+と∗ は左結合とする。

この例からわかるように，変換してもオペランド *a, b, c* の順序は変わらず，また変換後は後置きされた演算子のオペランドが一意に決まるので，括弧は不要である。一般に，識別子 id と単項演算子 uop と二項演算子 bop から構成される逆ポーランドの式の構文はつぎのように与えられる。

P ::= id | PP bop | P uop

算術式から逆ポーランド記法の式を求める関数 Polish はつぎのように与えられる。

Polish(uop *x*) = Polish(*x*) uop

Polish(*x* bop *y*) = Polish(*x*) Polish(*y*) bop

Polish((*x*)) = Polish(*x*)

Polish(id) = id

ここで，*x, y* は変数であり，任意の算術式を代入してよい。この関数 Polish を，上の例の最後の式 *a*+*b*∗(*b*+*c*)∗*c* に適用するとつぎのようになる。

Polish(*a*+*b*∗(*b*+*c*)∗*c*) = Polish(*a*) Polish(*b*∗(*b*+*c*)∗*c*)+

= *a* Polish(*b*∗(*b*+*c*)∗*c*)+

= *a* Polish(*b*∗(*b*+*c*)) Polish(*c*)∗+

= *a* Polish(*b*) Polish((*b*+*c*))∗Polish(*c*)∗+

= *a b* Polish((*b*+*c*))∗Polish(*c*)∗+

= *a b b c*+∗*c*∗+　(∵ Polish((*b*+*c*)) = *bc*+, Polish(*c*) = *c*)