目次

形式言語理論は, 自由モノイドの部分集合 $P(\Sigma^*)$ である言語を, 種々の形式文法 G を用いて分類する関手 $L : C(G) \to P(\Sigma^*)$ を定義し，性質を研究する学問である．形式文法 G に，特に特有な図示やメンタルモデルが存在する場合，抽象機械などと呼ばれる．テクノロジーとしての解釈，ミームとしての表現.

第 1 章

有限オートマトンと正則言語の理論

形式言語理論は，抽象機械というモデルによって，言語を分類し，包含階層 (hierarchy) を得る．これは計算複雑性クラスの分類に等価である．抽象機械はまるでモデルのように，形式言語のクラス構造を安定的に定義する役割を果たし，それ自体が形式科学的対象である．その際のモデル間の射のようなものが「還元（算譜）」である．この存在を combinatorical に構成して置けば，物理的に安定な概念が浮かび上がってくる．「計算可能性」はその一番大きなクラスである．

その中でも，有限オートマトンと正則言語の言葉だけでも，多くの問題は定式化出来る（PCP，古典的渡船問題など）．その他プッシュダウンオートマトンや Turing 機械などの多くの抽象機械 (automata) を伴う計算理論は，現実の問題を言語への帰属条件 $x \in L$ へと言い換えるから，特に数学の入り口となるものである．実際抽象機械と言語の理論は，Turing と Wittgenstein に date back することとなる．その他の計算モデルも本質的には同じだが，観念を符号化する記号と，その間の変化によって人のメンタルモデルが全て形式として得られるのは予想される，計算はこの意味でいくらか人間中心的な概念である．おそらく，高度な数学性に特徴付けられて，数学の上で夢想される計算機自然の法界であり，古典計算機も量子計算機もその物理学的な流出だと思える．私の一番好きな計算モデルである．

有限オートマトンはコンパイラの字句解析の部分，文脈自由言語はパーサー部分，

数学基礎論は形式言語についての理論である．あるいは数学自体がそうかもしれない．自然言語の自然性を飛び出して，もう一つの自然に手を伸ばそうとする言語が自然言語で，今は計算機としての豊かな実装をも持つ．

An automaton is a finite representation of a formal language that may be an infinite set.

すると，記号列を L と $\Sigma^* \setminus L$ の 2 つに分ける過程を計算と定義し，これを実行するオートマトン（Turing 機械）の存在性によって計算可能性を定義できる．

まずは，そのような抽象機械のうち小さなクラスである，有限状態機械を見る．有限のオートマトンには 2 つのクラスがあって，人間が描き易いのは非決定性オートマトンの方である．これは高級プログラミング言語と同じような原理であって，そこから決定性オートマトンに書き換える算譜が存在する．まるでコンパイラのように決定性オートマトンへと還元してくれる．特に ϵ-動作について拡張しても同様の還元の原理で等価になるが，これはさらに構成し易い．

1.1 言語演算：Kleene 閉包と連結と逆転

議論領域は有限生成される空間，関係による指定が言語，その関係を特性関数とみるか算譜とみるか

議論領域は毎度有限集合 Σ から生成される空間 Σ^* に決定される．するとこの議論領域はいつでも，

1. 文字列の連結についてモノイド $(\Sigma^*, \cdot, \epsilon)$ の構造を持っている．これが生成代数 $*$ に沿って，言語の空間 $P(\Sigma^*)$ にも連結の構造をいれる．
2.「零元を省くと群になる」のアナロジーだが，今回はならないので Σ^+ とでも書く．

この中で，言語 $L \in P(\Sigma^*)$ を指定する方法を考えたい．有限オートマトンの方法で指定できる言語は特に性質がよく，**正則言語**と呼ぶ．部分関数 $\mathbb{N} \nrightarrow \mathbb{N}$ を原始再帰的な方法で指定できるものを計算可能と呼ぶのと並行な議論だが，今回の方が真に小さいクラスである．また，問題の扱いも，形式言語理論では決定問題に限られる場合が多い．言語 $L = \{x \in \Sigma^* \mid P(x)\}$ を指定するのに述語 P を用いる方法と同様，算譜とは関係である．関数も関係である．

1.1.1　全体空間の措定：アルファベット

形式言語では，必ず基底たるアルファベットを指定して，それが生成する全体空間を指定してから議論が始まる．この時の帰納的な「生成」を単項演算と見て，Kleene star $*$ で表す（閉包と見て Kleene 閉包ともいう[a]）．「真の生成」を $+$ で表す．元々 Stephen Cole Kleene 09-94 がある種の automaton を特徴付けるために導入し，現在では正則表現にも残っている．Kleene 閉包の概念は，生成として Σ^* を定義してからは，**自由モノイド**として一般化されている．

[a] 基底的なところといい，閉包と生成の両軸が存在するところといい，とても位相っぽい．位相空間論は数学基礎論の一つの雛形になっているのだろうか

定義 1.1.1 (alphabet)**.** 空でない記号の有限集合 Σ を**アルファベット**という．

例 1.1.2 (アルファベットの例)**.**

1. $\Sigma = \{0, 1\}$．特に極まった形式科学で使う．
2. $\Sigma = \{a, b, c, \cdots, z\}$．自然言語でいう「アルファベット」．
3. 全てのアスキー記号の集合．ウェブブラウザにとってのアルファベット．
4. タンパク質のアルファベット $\Sigma = \{A, R, N, D, B, C, Q, E, Z, G, H, I, L, K, M, F, P, S, T, W, Y, V\}$．生物という計算機のアルファベット．
5. 計算機言語のアルファベット $\Sigma = \{\text{begin, if, end, for, while, do, else}\}$．この要素を token とも言う．
6. DNA のアルファベット $\Sigma = \{a, t, c, g\}$．遺伝子という計算機のアルファベット．

定義 1.1.3 (string / word というデータ構造)**.**

1. アルファベットの有限列 ${}^{<\omega}\Sigma = \cup_{n\in\mathbb{N}}\Sigma^n$ の元を**文字列**または**語**という．
2. 文字列 ω について，その長さ n を帰納的に定義し，$|\omega| := n$ と書く．
3. 文字列の i 番目の要素を，数列と見て x_i，文字列に作用する演算子と見て $x[i]$ などと書く．

注 1.1.4. 概念としても，長さ 1 の列とアルファベット 1 つとは違うし，実装も"1 Null"と"1"とで実体が違うことが多いだろう．

記法 1.1.5 (Kleene star)**.**

1. Σ^0 の元は空列 $\epsilon := [\,](:= \varnothing)$ のみである．空列は λ とも書く．
2. 記号を表すのは $a, b, c, \cdots$ で，記号列を表すのには $z, y, x, w, \cdots$ を使う．
3. アルファベット Σ からなる全ての有限列の集合を $\Sigma^* := {}^{<\omega}\Sigma$ とも書く．$\Sigma^+ := \Sigma^* \setminus \{\epsilon\}$ とする．

これは正規表現に通じる．発想は，Kleene star $*$ の中立元が 0 で，和 $+$ の中立元が 1 だというところだろうか．

1.1.2　文字列結合演算のモノイドとして

文字列の代数的操作として「文字列結合演算 $*$」を導入する．するとこれがうまくて，Kleene starΣ^* とは，この二項演算について閉じている最小の集合として特徴付けられる（Kleene 閉包）．こうして連結 $*$ の構造は言語の空間 $\mathrm{P}(\Sigma^*)$ にも持ち上がる．他にも，反転 ^{R} の代数的構造も定義でき，同じく持ち上がる．

定義 1.1.6 (concatenation, power)**.**

1. 列の**連結**を $xy := x \cdot y$ と書くこととする．すると，$(\Sigma^*, \cdot, \epsilon)$ はモノイドをなし，$(\Sigma^+, \cdot, \epsilon)$ は半群である．
2. また，**指数**を $x^0 = \epsilon, x^1 = x, x^2 = x \cdot x, \cdots$ と定義する．

定義 1.1.7 (prefix, sufflix, subword, subsequence)**.** アルファベット Σ による 2 つの記号列 $x, y \in \Sigma^*$ について，

1. $\exists u \in \Sigma^*, x = yu$ が成り立つ時，y を x の**接頭語**と言う．
2. $\exists u \in \Sigma^*, x = uy$ が成り立つ時，y を x の**接尾語**と言う．
3. $\exists u, w \in \Sigma^*, x = uyw$ が成り立つ時，y を x の**部分語**と言う．
4. $x = x_1 \cdots x_n$ と部分列 $1 \leqslant i_1 < \cdots < i_m \leqslant n$ が存在して，$y = x_{i_1} \cdots x_{i_m}$ を満たす y を，**部分系列**と言う．

定義 1.1.8 (reverse)**.** $x = x_1 \cdots x_n$ に対し，$x^R := x_n \cdots x_1$ とする．

1.1.3　言語

全体空間 Σ^* を「生成」の見方で用意し，「代数」の見方を滴下した．その部分集合を言語といい，これを分類することを考えるのが形式言語理論である．言語の空間 $\mathrm{P}(\Sigma^*)$ には通常の集合演算だけでなく，文字列演算から持ち上がった構造 $*, {}^*, {}^+, {}^R$ が定義される．

定義 1.1.9 (language)**.** アルファベット Σ に対して，部分集合 $L \subset \Sigma^*$ を**言語**という．多くは，Σ^* 上に帰納的定義により建設される．自然言語では，L の決め方，アルファベット Σ の決め方が生物学的な理由で発生する．

注 1.1.10. 一階論理の文脈でいう「言語」とは意味がずれる．この意味では，むしろ well-formed formula が一階論理の「言語」をなす．

例 1.1.11 (Post Correspondence Problem)**.** 「有限列が上下 1 組書かれた札（を表す文字列）が有限種類・各種類可算無限個与えられる．これらを有限枚並べて，上下の文字列を一致させることが出来るか？」

$$\mathrm{PCP} := \left\{ u_1 \# v_1 \# \cdots u_n \# v_n \;\middle|\; \begin{array}{c} u_i, v_i \in \{0,1\}^* \text{であり，ある列 } i_1, \cdots, i_m \in [n] \text{ が存在し，} \\ u_{i_1} \cdots u_{i_m} = v_{i_1} \cdots v_{i_m} \text{が成り立つ．} \end{array} \right\}$$

この問題を解く算譜が存在しないことを証明しようとすると，気が遠くなるほどに掴みどころがない．非停止性の認識問題（定理 3.6.11）は，巧妙な方法で最初の札の指定付きの Post の文字列揃え問題に帰着される．従って，PCP は決定不能である．

言語の上に，Σ^* から代数構造を入れる．

定義 1.1.12 (言語の積・反転・Kleene closure)**.** Σ 上の 2 つの言語 L, L' について，

1. (concatenation) $L_1 \cdot L_2 := \{xy \mid x \in L, y \in L'\}$ と定義する．
2. (power) $L^0 = \{\epsilon\}, L^1 = L, L^2 = LL, \cdots$ と定義する．
3. (Kleene closure) 言語 L の閉包を $L^* = \cup_{n \in \mathbb{N}} L^n, L^+ = \cup_{n=1,2,\cdots} L^n$ とする．
4. (reverse) 言語の反転を $L^R := \{x^R \mid x \in L\}$ とする．

注 1.1.13. この定義は，線型部分空間の和と全くパラレルな定義である．文字が形式的に等しくなければ，直積との同型が存在する．

以上より，言語を集合と見て，集合演算よりさらに詳細な演算

1. 連結 $\cdot, \cdot$ の通常の持ち上がり．
2. Kleene 閉包 *．$\cdot$ について閉じている空間を生成する生成子．
3. ${}^-$．
4. 逆転 ^{R}．

を定めた．

1.1.4　言語を問題と見る約束

通常問題は部分関数 $\mathbb{N} \nrightarrow \mathbb{N}$ としてコードされるが，これをさらに形式科学的に還元して，$\Sigma^* \to \Sigma$，さらには $\{0,1\}^* \to \{0,1\} \simeq \mathrm{TV}$ とする．この還元された yes, no を答えさせる，形式言語理論的な問題観念を**決定問題**という．
このクラスの決定問題は理論的に扱いやすいだけでなく，例えば，C 言語のコンパイラが現実的な時間内で快適に使用可能なものが作れるかどうかの問題と「還元 (reduction)」の関係にある．この還元という考え方があるから，「帰属問題」というクラスに限って良いのである．計算量理論に自然数上の部分関数を考えれば済む理由（**算術化**）と同じである．
これは所属関係 $x \in X$ の特別な場合である（X が Kleene star $*$ で有限生成される場合）．この有限生成性が，言語の本質を作る．何のアルファベットから有限生成するかは，使用者に依る．使用者に所有されない言語本来の形式に宿る性質をつかみたい．最終的には 0, 1 と自然数に集約されることとなる（第 3.6 節）．

> 「問題」を言語と定義することは，計算量の理論で重要な問題を扱う妥当な方法として，時間の検証にも耐えてきた．この理論では，ある問題の計算量の下限を証明することに興味がある．特に重要なのは，ある問題が入力の指数関数よりも少ない時間量で問題が解けないことを証明するための技法である．はい／いいえを答えさせるという言語型の問題は，「これを解け」という意味での問題と，よく知られている多くの場合に「同程度に難しい」ことがわかっている．

定義 1.1.14 (problem)**.** オートマトン理論で**問題** L とは，与えられたアルファベット Σ の文字列 $w \in \Sigma^*$ に対して $w \in L$ かどうかを決定するクラスの「決定問題」を指す．

例 1.1.15 (素数判定問題)**.** 素数判定問題は，アルファベット $\Sigma = \{0,1\}$ の上に，言語 L_p として得られる．$11101 \in L_p$ である．この問題は数学的には解けてなくて，現実的なアルゴリズムは……？どうなんだ？

例 1.1.16 (SAT: Satisfiability problem)**.** 与えられた一つの命題論理式に対して，それを充足させる付値が存在するか．NP 完全である．即ち，NP（非決定性 Turing 機械によって多項式時間で解くことができる問題）かつ NP 困難．これは Stephan Cook の 1971 年の論文 "The Complexity of Theorem Proving Procedures"で最初に示され，Cook-Levin's theorem と呼ばれている．その論文では計算機科学最大の問題である「P 対 NP 問題」も定式化している．

例 1.1.17 (the Entscheidungsproblem)**.** 「一階述語論理の文が与えられたとき，tautology（＝証明可能）かどうか判定するアルゴリズムはあるか？」これが可能なら，非停止性の認識が出来てしまうので，これも否定される．

1.2　有限状態機械：NFA と DFA

NFA と DFA

有限オートマトンのうち，特に数学モデルとして適切な 2 つ

1. 非決定性有限オートマトン (NFA)
2. 決定性有限オートマトン (DFA)（NFA の特別なクラス）

を定義し，DFA のなす圏 Aut を構成する．今回は DFA を形式的な対象として定義した．この抽象機械と呼べる対象の射は，「大は小を兼ねる」という性質を持たせたいから，模倣算譜の存在性で定義する．
非決定性は推論能力の源泉とも考えられる．決定性は圏をなし，論理的な完結性を持つ．これは不思議な双対なのかもしれない．
なお，これは退化した概念で，出力と終了状態の 2 つを分離することも可能である．出力アルファベット Δ と出力関数 $\gamma : Q \times \Sigma \to \Delta$ の追加は等価な抽象機械を定義し，これを順序機械という．

1.2.1　有限状態機械とその圏

定義 1.2.1 (NFA: nondeterministic finite automaton, DFA)**.** **非決定性有限オートマトン**とは，次の条件を満たす 5-組 $M = (Q, \Sigma, \delta, q_0, F)$ のことである．

1. Q：状態からなる空でない有限集合．
2. Σ：入力アルファベットからなる集合．
3. $q_0 \in Q$：初期状態．
4. $F \subset Q$：受理状態の集合．
5. $\delta \subset Q \times \Sigma \times Q$：**状態遷移関係**と呼ばれる 3 項関係．[†1]
   (i) 要素 $(p, a, q) \in \delta$ を遷移と呼び，$p \xrightarrow{a} q$ と表す．
   (ii) 状態遷移関係 δ が，$\forall p \in Q,\ \forall a \in \Sigma,\ \exists! q \in Q,\ (p, a, q) \in \delta$ を満たすとき，このオートマトン M を**決定的**であると言う．
   (iii) このとき，$\delta \in \mathrm{Map}(Q \times \Sigma, Q)$ となるので，特に**状態遷移関数**と呼び，決定性の有限オートマトンを特に**有限オートマトン**と言う．

定義 1.2.2 (オートマトンの射：simulate)**.** 2 つの DFA $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1), M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ の間に次の 3 条件を満たす全射 $\varphi : M_1 \to M_2$ が存在する時，M_1 は M_2 **を模倣する**という．

1. 初期状態を保つ：$\varphi(q_1) = \varphi(q_2)$.
2. 停止状態を保つ：$\varphi^{-1}(F_2) = F_1$.
3. 遷移を保つ：$\forall a \in \Sigma, \forall q \in Q_1,\ \delta_2(\varphi(q), a) = \varphi(\delta_1(q, a))$，即ち次の図式が可換である．

$$\begin{array}{ccc} Q_1 \times \Sigma & \xrightarrow{\delta_1} & Q_1 \\ {\scriptstyle \varphi \times 1}\downarrow & & \downarrow{\scriptstyle \varphi} \\ Q_2 \times \Sigma & \xrightarrow{\delta_2} & Q_2 \end{array}$$

DFA（あるいは決定的な Moore 状態機械）がこの射についてなす圏を Aut と表す．

定義 1.2.3 (accepting computation, accepting language / 定める言語)**.**

1. 有限状態機械 M が記号列 $w = a_0 \cdots a_n$ を受理するとは，遷移の系列
$$q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} \cdots \xrightarrow{a_n} q_{n+1} \quad (q_{n+1} \in F)$$
が存在することをいう．この系列を**受理計算**と言う．2 で定義する記号を使えば $w \in L(M)$ である．
2. 次元を一つあげる．$L(M) := \{w \in \Sigma^* \mid M \text{ は } w \text{ を受理する}\}$ を**受理言語**といい，M が集合 L を受理すると言った時は $L = L(M)$ であることをいう．

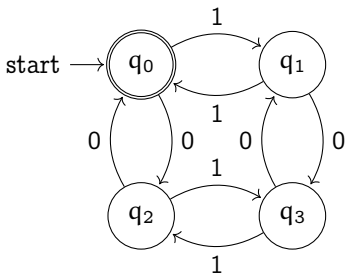
注 1.2.4.

1. $L \subset L(M)$ ではなく，意味が非常に強いことに注意．オートマトンを言語と対応させる写像 $L : \mathrm{Aut} \to P(\Sigma^*)$ を定めるためである．
2. 非決定性オートマトンは実際には受理し損ねることがあり得る．しかし，「受理する可能性がある」ものでなければならない．また受理計算が 1 つである必要もない．
3. $q_0 \in F$ であるとき，このオートマトン M は空列 ϵ を受理する．
4. オートマトンの持つ遷移 δ は非常に圏的である．これを視覚化するオートマトン独自の図式 (diagram)・**状態遷移図**がある．また，Turing 機械の時と同様，遷移表を与えることでも有限オートマトンを与えることと等価である．
5. 特に，オートマトンを計算機上で実装するための言語としては，統一モデリング言語 (Unified Modeling Language) や仕様及び記述言語 (Specification and Description Language) がある．

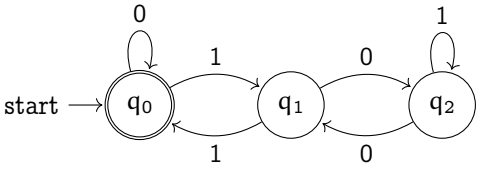
[†1] 従って，部分関数 $\delta : Q \times \Sigma \to Q$ でも良い．

例 1.2.5.

1. $\{x \in \{0,1\}^* \mid x$ は $0,1$ を偶数個含む $\}$ を受理するオートマトンは，1 が奇数個を q_1，0 が奇数個を q_2，どちらも奇数個を q_3 とすれば次のように 4 状態を用意すると良い．



2. $\{x \in \{0,1\}^* \mid [x]$ は 3 の倍数 $\}$ を受理するオートマトンは，$[x] \equiv 0 \mod 3$ の状態で 1 を受理すると $[x1] \equiv 1 \mod 3$，0 を受理すると $[x0] \equiv 0 \mod 3$ というように考えていくと，次の 3 状態オートマトンで受理できる．



1.2.2　代数と余代数

signature

指標とは，極めて数学基礎論的な概念である．これを用いて，代数系を Ω-代数として一般化できる．
F-代数とは，Ω-代数の圏論的な言い換えである．私が無意識にやっていたことである．すると，代数規則は関手 F の言葉を用いて貼り合わせられ，一つにまとまる．結局 signiture[a]とは関手のことだと思える．群，束などもこの言葉で全く並行に一般化される．
この観念を通じて，計算機科学の種々の概念は代数化できる．データ構造は（始）代数を定め，オートマトンも余代数として等価に理解できる．これを計算機科学の分野に圏論を通じて輸入したものを，特に F-余代数という．

[a] プログラミングで、メソッドや関数の、名前および引数の数や型の順序などの組み合わせ。戻り値の型を含む場合もある。

定義 1.2.6 (Ω-algebra).

1. 集合 Ω を（代数的）**指標 (signature)** といい，非論理記号（演算子記号や述語記号）の集合とする．
2. $ar : \Omega \to \mathbb{N}$ を arity という．
3. 指標 Ω が定める **Ω-代数** A とは，carrier $|A|$ と，各指標 $\omega \in \Omega$ の解釈としての関数 $a_\omega : |A|^{ar(\omega)} \to |A|$ の組 $A = (|A|, (\omega)_{\omega\in\Omega})$ である．
4. Ω-代数の射 $h : A \to B$ とは，写像 $h : A \to B$ であって，任意の演算子 $\omega \in \Omega$ について $h(a_\omega(x_1, \cdots, x_{ar(\omega)})) = b_\omega(h(x_1), \cdots, h(x_{ar(\omega)}))$ を満たすもののことをいう．
5. こうして Ω-代数のなす圏を Ω-Alg と表す．

定義 1.2.7 (signature). 指標の概念は組 (Ω, ar) としたが（これは代数的指標と呼ばれる），より精緻になる．非論理記号は基本 2 種類に分類できる．[†2] 指標 Σ とは，次の 3-組 $(S, Rel(\Sigma), Func(\Sigma))$ のことである．

1. S は型 (type, sort) の集合である．
2. $Rel(\Sigma)$ は関係記号の集合で，関数 $ar : Rel(\Sigma) \to S^*$ が定まっている．
3. $Func(\Sigma)$ は関数記号の集合で，関数 $(dom, cod) : Func(\Sigma) \to S^* \times S$ が定まっている．

[†2] https://ncatlab.org/nlab/show/signature+(in+logic)

注 1.2.8.

1. ほとんどの数学概念は $|S| = 1$ の指標で記述される．これを **single-sorted signature** という．
2. 特に大きな multisorted な指標は圏やグラフの定義である．ここで斎藤先生はあの定義を持ち出した．
3. single-sorted である時，自由モノイド S^* は $\mathbb{N}$ と同型であるから，関数 ar は通常の意味での arity である．
4. 従って，関係と関数の区別は，n-項関係，n-項演算と見た時，dom を見ているのかである．0-ary の関数のことを定数という．
5. $\mathrm{Func}(\Sigma) = \emptyset$ の時，これを **relational signature** という．
6. $\mathrm{Rel}(\Sigma) = \emptyset$ の時，これを **equational** または **algebraic** signature という．ただし，記号 $=$ は普通論理記号とみなす．

定義 1.2.9. $\{x, y, z, \cdots\}$ を変数からなる集合とし，E を言語 $\{x, y, z, \cdots\} \cup \Omega$ 上の方程式の集合とする．E を充すような Ω-代数は再び圏をなし，これを (Ω, E)-Alg と表す．

この極めて形式言語的な概念「指標 Ω」は，結局，直和の言葉によって，自己関手 F 1 つにまとまる．Ω-代数の圏と F-代数の圏は等しい．

定義 1.2.10 (F-algebra, F-coalgebra)**.**

1. 圏 C とその上の自己関手 $F : C \to C$ について，C の対象 $A \in C$ とその射 $f : F(A) \to A$ の組 (A,f) のことを **F-代数**という．A をこの代数の **carrier** という．この双対概念を **F-余代数**という．即ち，射は $f : A \to F(A)$ の向き．
2. 余代数の射 $(A, f) \to (B, g)$ とは，C の射 $\alpha : A \to B$ であって，次の図式を可換にするもののことである：

$$\begin{array}{ccc} A & \xrightarrow{f} & F(A) \\ {\scriptstyle\alpha}\downarrow & & \downarrow{\scriptstyle F(\alpha)} \\ B & \xrightarrow{g} & F(B) \end{array}$$

3. これにより，F-余代数は圏をなす．F-代数も同様であり，これを **variety** という．
4. F-代数の圏が始対象を持つとき，特に **F-始代数** (initial F-algebra) と呼ぶ．この双対概念を **F-終余代数** (Terminal F-coalgebra) という．

例 1.2.11 (始代数の例)**.** プログラミングで使われるリストや木構造のようないくつもの有限データ構造が、特定の自己関手の始代数として得られる。**始対象と，それが生息する圏上の関手こそが，帰納や再帰といったものの一般の枠組みを与えるものだったのである.**

1. 自然数とは，集合の圏 Set 上の自己関手 $F = 1 + -$ についての F-代数 $(\mathbb{N}, 0 + \mathrm{succ})$ である．Set は始対象 0 を持つので，これは始代数である．[†3]
2. Set の自己関手 $1 + \mathbb{N} \times -$ を考えると，集合 $X \in$ Set に対して，始代数 $(X, [x, f])$ を定める．この場合の始代数は、自然数を要素とする有限な長さのリスト全体の成す集合、その点としての空リストおよび自己写像 cons（与えられた自然数と有限リストから、その自然数をリストの先頭に付け加えてえられるリストを返す函数）の組で与えられる。[†4]

例 1.2.12 (余代数の例)**.** 余代数は状態をもつシステム (状態遷移系や、オブジェクト指向プログラミングにおけるクラスなど) や、無限の内容を持ちうるデータ構造 (ストリームなど) などの挙動を、十分に一般的かつ利用しやすい形で記述できることから、計算機科学で広く用いられるようになった。代数的仕様がシステムの動作を関数として (特に、コンストラクタによって生成される帰納的なデータ型を用いて) 記述するのに対し、余代数的仕様はシステムの動作を余帰納的なプロセス、つまりセレクタの出力によって観測される内容として (オートマトン理論のような考え方で) 記述する。このときありえる全ての無限動作を漏れなく重複なく集めてきた集合が終余代数となるため、終余代数も重要な役割を果たす。余代数によって記述されるシステムの性質を記述するのには、余代数的様相論理が適している。[†5]

1. p 進整数も距離空間として，終余代数として特徴付けられる．[†6]

[†3] https://en.wikipedia.org/wiki/F-algebra
[†4] https://ja.wikipedia.org/wiki/始代数
[†5] https://ja.wikipedia.org/wiki/F 余代数
[†6] Prasit Bhattacharya, The p-adic integers as final coalgebra,https://arxiv.org/abs/1504.01408

定義 1.2.13 (遷移関数の currying による，余代数によるオートマトンの定義)．$\delta : Q \times \Sigma \to Q$ から，$\delta : Q \to Q^{\Sigma}$ を定める．ここで，終状態 $F \subset Q$ を，特性関数 $\chi_F : Q \to 2$ とみなすと，$\alpha := (\delta, \chi_F) : Q \to Q^{\Sigma} \times 2$ を定め，これは H-余代数 (Q, α) を定める．ただし，これを定める Set 上の自己関手 H は，$H(-) = -^{\Sigma} \times 2$ というものである．

注 1.2.14 (２つの定義の等価性)．こうして定めたオートマトンの余代数 (Q, α) の射 $: f(Q, \alpha) \to (Q', \alpha')$ s.t. $\alpha' \circ f = H(f) \circ \alpha$ かつ $\delta'(\sigma, f(q)) = f(\delta(\sigma, q))$ かつ $f(F) = F'$．即ち，simulation としてのオートマトンの射の概念と完全に一致する．

この圏の終対象 (T, α_T) がオートマトンとして面白い？また，non-deterministic の場合も同様に余代数化出来る．

1.2.3　決定性有限オートマトン論

決定性有限オートマトンは言語を定めることを定義したが，形式言語理論の本領は，DFA 自身を研究対象とすることである．DFA の本質は，意味論を剥ぎ取れば，算譜 δ である．これだけが言語 $L(M)$ を定めていることを見る（$L(M)$ の δ^* による特徴付け）．

定義 1.2.15 (遷移関数が，文字列の集合上に定める関数)．有限オートマトン $(Q, \Sigma, \delta, q_0, F)$ の遷移関数 $\delta : Q \times \Sigma^* \to Q$ を，次のように，$\Sigma \to \Sigma^*$ の構成に関する帰納法により $q^* : Q \times \Sigma^*$ に拡張する．

1. $\delta^*(q, \epsilon) = q \quad \forall q \in Q$.
2. $\delta^*(q, xa) = \delta(\delta^*(q, x), a) \quad \forall q \in Q, x \in \Sigma^*, a \in \Sigma$.

命題 1.2.16 (M が x を受理することの特徴付け)．次の 2 条件は同値．

1. M は x を受理する：$x \in M$.
2. $\delta^*(q_0, x) \in F$.

命題 1.2.17. 有限オートマトン M と記号列 $x, y \in \Sigma^*$ に対して，$\delta^*(q, xy) = \delta^*(\delta^*(q, x), y)$.

注 1.2.18 (非決定性オートマトンの言語：別定義)．非決定性オートマトン $M = (Q, \Sigma, \delta, q_0, F)$ も，$\delta \subset Q \times \Sigma \times Q$ を $\overline{\delta} : Q \times \Sigma \to P(Q)$ と同一視すれば，構造的帰納法によって $\delta^* : Q \times \Sigma^* \to P(Q)$ が定まる．そして，非決定性オートマトン M が定める言語は，

$$L(M) := \{w \in \Sigma^* \mid \delta(q_0, w) \cap F \neq \emptyset\}$$

と定める．つまり，必ずしも 100% の確率で受理状態に到達しなくても良い．

このように非決定性オートマトンの遷移関係 $\delta \subset Q \times \Sigma \times Q$ を冪集合上への写像と見なす手法は，NFA→DFA の還元算譜である subset construction の世界観と適合する．

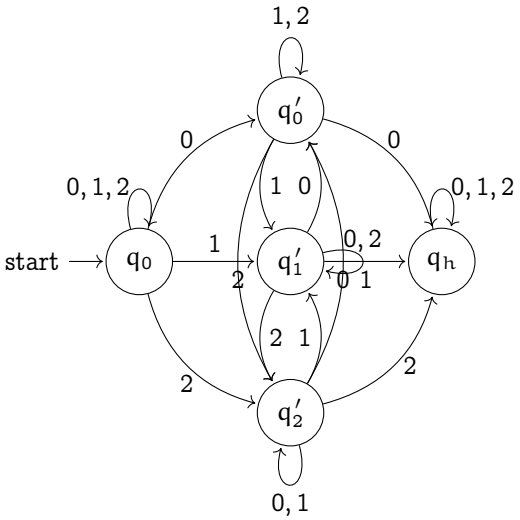
1.2.4　オートマトンの標準形

オートマトンの同値類を作って，うまく代表系を取る算譜を考えたい．このような志向を持った理論を標準形の理論という．

1. 到達不能な状態の剪定：「受理計算」の概念を一般化し，任意の 2 状態間に「到達可能」の二項関係を定める．

例 1.2.19 (非決定性オートマトン：非常に自由度が高く，構成しやすい)．非決定性オートマトンは，受理計算を左右させない範囲で（特に自己遷移を）遷移を増やせる．

1. 次のオートマトンは，最終文字が必ず以前出てきた文字になる文字列がなす言語 $\{a_1 a_2 \cdots a_n \in \{0, 1, 2\}^* \mid n \geqslant 2, a_n \in \{a_1, \cdots, a_{n-1}\}\}$ を受理するが，q_0', q_1', q_2' の自己遷移は追加しても追加しなくても良い，受理計算に影響を与えないからである！



2. 次のオートマトンは $\{a_1a_2\cdots a_n \in \{0,1,2\}^* \mid n \geqslant 2, a_n \notin \{a_1,\cdots,a_{n-1}\}\}$

定義 1.2.20 (accessible)．2 つの状態 $p, p' \in M$ について，p から p' に**到達可能**であるとは，記号列 $x = a_0\cdots a_n$ と，それが定める遷移系列 $p \xrightarrow{a_0} p_1 \xrightarrow{a_1} \cdots \xrightarrow{a_n} p'$ が存在することをいう．

命題 1.2.21 (有限オートマトンの標準形：初期状態から生成される部分について整形)．M に対して，次のように定めるオートマトン M' について，$L(M) = L(M')$．また，M が決定的ならば M' も決定的である．

1. 初期状態 p_0 から到達可能でない状態を取り除く．
2. それらを端点とする遷移を取り除く．

1.2.5　非決定性オートマトンの応用

応用例 1.2.22 (egrep, fgrep)．非決定性オートマトンはテキスト検索にも使われる．UNIX コマンド egrep, fgrep では，次の 2 つを混ぜて実装されている．

1. NFA を模倣するプログラムを書く．
2. subset construction で DFA に還元し，それを模倣するプログラムを書く．

このクラスの問題に対する subset construction では，状態数の増加が伴わないことが証明できる．

1.3　決定性オートマトンへの還元算譜と正則言語：決定性に依らない安定な構造

NFA と DFA

有限オートマトンにおいて決定性／非決定性の別はそれが定める受理言語の観点からは対称であることが分かる．なぜなら，その間には subset construction という還元算譜が存在するのである．(まるで座標変換)．決定性の違いは，実装としては大きな違いに思えるが，形式的対象としては煩瑣な違いであることがわかる．
これは新たな記号 a を認識する度に，非決定性オートマトンの遷移し得る範囲を全て抽出して，その情報を部分集合 $S \subset Q$ として格納していく．このような動きをする決定的オートマトンを冪集合構成の上に展開してしまうまさに数学基礎論的方法である．これがうまくいく理由は，非決定性オートマトンの「遷移しうる範囲」は決定的に決まっていることに起因する．所詮自動人形なのである．そして「動きうる範囲」が F と共通部分を持つようになれば，受理計算が存在することと等価になる．この構成を通じて，新たな語に対する受理計算が発生することもない．
こうして，言語の圏 $P(\Sigma^*)$ の中に有限オートマトンが定めるクラス $L(\mathrm{Aut})$ を，正則言語という．これは形式言語のクラスの中で，一番小さい部類のクラスとなる．

定理 1.3.1 (非決定性有限オートマトンと決定性有限オートマトンの等価性)．言語 $L \in P(\Sigma^*)$ に対して，次の 2 条件は同値．

1. L は非決定性有限オートンマトンによって受理される．
2. L は決定性有限オートマトンによって受理される．

［証明］．定義より決定性オートマトンは非決定性オートマトンの特殊な場合としたから（$\delta : Q \times \Sigma \to Q$ は三項関係の特別な場合に他ならない，あるいは初期状態に余分な自己遷移を加えることによっても），2.⇒1. は成り立つから，⇐ を示す．

非決定性オートマトン $M = (Q, \Sigma, \delta, q_0, F)$ から次のように構成した (subset construction) 決定性オートマトン $\tilde{M} = (\tilde{Q}, \Sigma, \tilde{\delta}, \tilde{q}_0, \tilde{F})$ は，同じ言語を受理する：$L(M) = L(\tilde{M})$ と示す．

1. $\tilde{Q} := P(Q)$.
2. $\tilde{q}_0 := \{q_0\}$.
3. $\tilde{F} := \{S \in P(Q) \mid S \cap F \neq \emptyset\}$.
4. $$\begin{array}{ccc} \tilde{Q} \times \Sigma & \xrightarrow{\quad delta \quad} & \tilde{Q} \\ \cup & & \cup \\ (S, a) & \longmapsto & \{q \in Q \mid \exists p \in S,\ (p, a, q) \in \delta\} \end{array}$$

まず，$L(M) \subset L(\tilde{M})$ を示す．任意に $w = a_1 \cdots a_n \in L(M)$ を取ると，Q 上の受理計算

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} q_n$$

が存在する．すると，$S_0 = \{q_0\}, S_i = \{delta(s_{i-1}, a_i)\}$ とした系列を考えると，構成法より $q_n \in S_n$ であるから，$S_n \in F_n$ であり，受理計算が構成できたことになる．よって，$w \in L(\tilde{M})$ である．

次に，$L(M) \supset L(\tilde{M})$ を示す．全く同様に，$w \in L(\tilde{M})$ の受理計算

$$S'_0 \xrightarrow{a_1} S'_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} S'_n$$

が $\tilde{M}$ に存在する時，$q_n \in S'_n$ から逆算して，q_0 からの M の系列が作れる．

よって，上の構成によって，決定性オートマトン $\tilde{M}$ について $L(M) = L(\tilde{M})$. □

注 1.3.2. 非決定性オートマトンの余代数を，冪集合構成によって無理やりまとめているのが $\tilde{\delta}$ である！

定義 1.3.3 (regular set / language)**.** （決定性）オートマトンによって受理される集合を**正則集合**または**正則言語**という．即ち，関手（？）$L : Aut \to P(\Sigma^*)$ の値域である．

1.4　ϵ 動作付き非決定性有限オートマトン

ϵ 動作付き非決定性オートマトンを Aut に引き込む

非決定性オートマトンは subset construction によって Aut に引き込まれた．ϵ 動作付き非決定性オートマトンも同様である．

ϵ 動作付き非決定性有限オートマトンはさらに人間がプログラムし易い高級言語となっている．ϵ 遷移の除去算譜も，ϵ 遷移について閉包を取るという step が増えるだけで本質的には subset construction の延長に過ぎない．従ってこれが定め得る言語も，正則言語に一致する．

定義 1.4.1 (ϵ-transition, ϵ-NFA)**.**

1. 空語 ϵ を読んで行う遷移を **ϵ-遷移**という．
2. 非決定性オートマトンの状態遷移関係 $\delta \subset Q \times \Sigma \times Q$ を，$\delta \subset Q \times (\Sigma \cup \{\epsilon\}) \times Q$ としたものを，**ϵ-動作付き非決定性有限オートマトン**という．

定義 1.4.2 (ϵ-closure)**.** ϵ-閉包 $ECLOSE(q) \subset Q$ を次のように機能的に定義する．

1. $q \in ECLOSE(q)$.
2. $p_1 \in ECLOSE(q)$ とする．$\exists a \in \Sigma, (q_1, a, q_2) \in \delta$（または $q_2 \in \delta(p_1)$）ならば，$q_2 \in ECLOSE(q)$.

定義 1.4.3 (遷移関数の拡張と，ϵ 動作付き非決定性有限オートマトンが定める言語)**.** $\delta^* : Q \times (\Sigma \cup \{\epsilon\})^* \times Q$ を次のように帰納的に定義する．

1. $\delta^*(q, \epsilon) = \mathrm{ECLOSE}(q)$.
2.

定理 1.4.4 (等価原理と ϵ 遷移の除去算譜)**.** 言語 $L \in P(\Sigma^*)$ に対して，次の 2 条件は同値．

1. L は ϵ 動作付き非決定性有限オートンマトンによって受理される．
2. L は決定性有限オートマトンによって受理される．

［**証明**］**.** 定義上 (2)⇒(1) は成り立つから，(1)⇒(2) を示す．
ϵ-動作付き非決定性有限オートマトン $M = (Q, \Sigma, \delta, q_0, F)$ が L を受理するとする．ϵ-動作についての閉包

$$\begin{array}{ccc} \widehat{\ } : P(Q) & \longrightarrow & P(Q) \\ \cup & & \cup \\ S & \longmapsto & \hat{S} := \left\{ q \in Q \;\middle|\; \begin{array}{l} \exists p \in S,\ \exists p_1, \cdots, p_n \in Q, \\ (p, \epsilon, p_1), \cdots, (p_n, \epsilon, q) \in \delta \end{array} \right\} \end{array}$$

と，状態の集合 S から記号 a を受理することで遷移し得る範囲を

$$\begin{array}{ccc} P(Q) \times \Sigma & \longrightarrow & P(Q) \\ \cup & & \cup \\ (S, a) & \longmapsto & S(a) := \{q \in Q \mid \exists p \in S,\ (p, a, q) \in \delta\} \end{array}$$

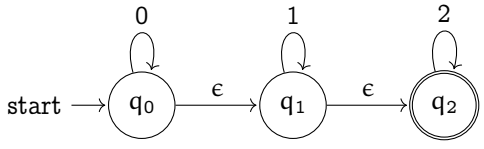
と定める．決定性オートマトン $\hat{M} = (\hat{Q}, \Sigma, \hat{\delta}, \hat{q}_0, \hat{F})$ を次のように構成すると，$L = L(\hat{M})$ である．

1. $\hat{Q} := P(Q)$.
2. $\hat{q}_0 := \widehat{\{q_0\}}$.
3. $\hat{F} := \{S \in P(Q) \mid F \cap S \neq \emptyset\}$.
4. $\hat{\mathrm{delta}} : \hat{Q} \times \Sigma \to \hat{Q}$ は $\hat{\mathrm{delta}}(S, a) = \widehat{S(a)}$.

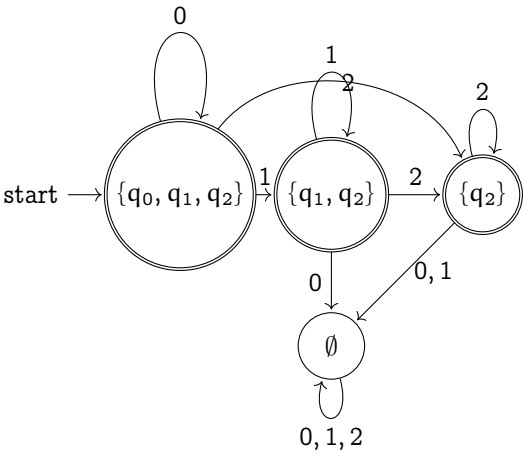
□

例 1.4.5 (ϵ 遷移の除去算譜)**.** 次の ϵ-遷移付き非決定性有限オートマトンには，0121 を受理する受理計算は存在しない．

	入力			
状態	0	1	2	ϵ
q_0	$\{q_0\}$	$\emptyset$	$\emptyset$	$\{q_1\}$
q_1	$\emptyset$	$\{q_1\}$	$\emptyset$	$\{q_2\}$
q_2	$\emptyset$	$\emptyset$	$\{q_2\}$	$\emptyset$



これを DFA に変換すると，最終墓地が明白になる．



1.4.1　ϵ-遷移付き非決定性有限オートマトンの応用

この新たな武器 ϵ 遷移と等価原理によって，次の正則言語の性質が簡単にわかる．

命題 1.4.6 (有限な言語は全て正則)**.** 任意の input alphabet Σ について，任意の有限部分集合 $L \subset \Sigma^*$ は正則言語である．

［証明］. ϵ-動作付き非決定性有限オートマトンを構成する算譜があるはずである． □

無限状態機械

以上の有限状態機械は，無限にも定義を拡張でき，同様の定理を証明できる．

1.5　正則言語の反復補題

正則言語の必要条件を考えたい．記号列の無限集合を認識する有限オートマトンは，必ずループを持つ．この正則言語（や特に文脈自由言語，定理 2.4.1）の必要条件を，反復補題という．
本質は同様に，「形式言語は所詮有限生成される模様であるから，ある閾値があってそれを超える範囲では有限性由来の反復模様がある」ということである．確かに Turing pattern っぽい．初出はこれ [6]. [a]

[a] 反復補題（英: Pumping lemma）とは、計算可能性理論において、あるクラスの形式言語に反復を施してもそのクラスに依然として属することを示すものである。ここでいう「反復」とは、その言語に含まれる十分に長い文字列が部分に分割可能で、その一部分を繰り返したさらに長い文字列も同じ言語に含まれるようにすることである。この補題の証明には、鳩の巣原理のような組合せ数学が必要とされる。反復補題の重要な具体例として、正規言語の反復補題と文脈自由言語の反復補題がある。文脈自由言語の反復補題の一種として、オグデンの補題もある。これらの補題は、ある言語が特定の言語クラスに属さないことを示すのに使われる。しかし逆に、反復補題を満たすことは必要条件ではあっても十分条件ではないので、ある言語があるクラスに属することを示すのには使えない。

定理 1.5.1 (Pumping Lemma)**.** $L \subset \Sigma^*$ を正則言語とする．このとき，ある定数 $N \geqslant 1$ が存在して，長さが N 以上の記号列 $x \in L \wedge |x| \geqslant N$ ならば，その分解 $x = uvw$ であって，

1. $1 \leqslant |v| < N$（これよりも粗い条件である $(1 \leqslant |v|) \wedge (1 \leqslant |uv| \leqslant N)$ としても示せる（定理 2.4.1））,
2. $\forall m \in \mathbb{N},\ uv^m w \in L$,

の 2 条件を満たすものが存在する．

［証明］. L は正則言語であるから，これが定める決定性有限オートマトンを $M = (Q, \Sigma, \delta, q_0, F)$ とする．$N = |Q|$ とすれば，$\{q_0\} \subset Q$ より $N \geqslant 1$ を満たす．$|x| =: n \geqslant N$ を満たす $x = a_1 a_2 \cdots a_n \in L$ を任意に取る．すると x が定める受理計算が存在するが，その系列の長さは $|Q|$ を超えているから，鳩の巣原理より同じ状態が少なくとも 2 回現れる：$\exists i, j \in n+1,\ (0 <) j - i \leqslant N \wedge p_i = p_j$．これについて，$u := a_1 a_2 \cdots a_i, v := a_{i+1} \cdots a_j, w := a_{j+1} \cdots a_n \in L$ とすると，v は

$1 \leqslant |v| < N$ で，状態 $p_i = p_j$ を中心としたループである．従って，任意の $m \in \mathbb{N}$ に対して $uv^m w \in L$ にも受理計算があるから，$uv^m w \in L(M)$. □

所感 1.5.2. 定理 2.4.1 の右線型文法 $A \to w \mid wA$ における場合の結果である．

系 1.5.3. 言語 $L = \{0^n 1^n \mid n \in \mathbb{N}\}$ は正則ではない．

［証明］．L を正則とする．すると，$N > 0$ が存在して，任意の $|x| =: n \geqslant N$ を満たす文字列 $x \in L$ について，$1 \leqslant |v| < N$ を満たす分解 $x = uvw$ が存在して，$\forall m \in \mathbb{N}$, $uv^m w \in L$ を満たす．いま，$x = 0^N 1^N$ とする．$|v| < N$ より，$v \in \{0\}^*$ または $v \in \{1\}^*$ または $v \in \{0,1\}^*$ である．最初の 2 つの場合は $m = 0$ について矛盾．最後の場合は $m = 2$ について，位置関係が逆転している 0, 1 が存在するので矛盾． □

系 1.5.4 (回文)**.** 言語 $L = \{ww^R \mid w \in \{0,1\}^*\}$ は正則ではない．

［証明］．L を正則とする．$1 \leqslant |v| < N$ であるので，$m = 0$ とすると回文ではなくなる． □

1.6　正則表現

正則言語 hack：適宜適切な代数的手法で有限生成して正則言語を得る代数的表現

以前の章で，Aut から言語 Σ^* のクラス：正則言語を定め，有限オートマトン 3 種の等価性をみた．ここで，抽象機械とは違って，正則言語に対する代数的記法を 1 つ導入する．これは実用化されているほど強力な代数的手法で，生成の概念を捉えている 1 つの形式言語である．正則言語にとってはメタ言語である．これは正則文法の強力な表現メディアの一つであろう（定義 2.6.1）．

定義 1.6.1 (regular expression / regex)**. アルファベット Σ 上の正則表現**とは，アルファベット $\Sigma \cup \{*, +,), (, \emptyset\}$ 上の記号列 $\mathrm{REGEX}_\Sigma \subset (\Sigma \cup \{*, +,), (, \emptyset\})^*$ であって，次のように帰納的に定義されるものである．正規表現を，それが表す正則言語へと対応させる解釈写像 $L : \mathrm{REGEX}_\Sigma \to \mathfrak{P}(\Sigma^*)$ が定まる．言語 $L(r) \subset \Sigma^*$ と正則表現 r を混同することも多い．

1. 空列を表す記号 $\emptyset$ は正則表現で，空な言語を表す：$L(\emptyset) = \emptyset$.
2. 各アルファベット $a \in \Sigma$ は正則表現であり，語 a と解釈される：$L(a) = \{a\}$.
3. r, s を正則表現とすると，$(r + s), (rs), (r^*)$ は正則表現であり，次のように解釈される：$L(r + s) = L(r) \cup L(s), L(rs) = L(r)L(s), L(r^*) = L(r)^*$．ただし列または言語について $L_1 L_2$ とは記号列の連結を表す．
4. 以上のように定義されるもののみが正則表現である．

所感 1.6.2. 正則表現は，文字列演算を引き継いでいる（定義 1.1.6）．適宜，適切な代数的手法で有限生成すれば良いという算段である．和集合と，連結と，Kleene 生成である．正則言語の閉包性（1.8 節，定理 1.8.2）が背景である．

例 1.6.3.

1. 実は集合としての積に相当するものは，正則言語については自然な発想だが，正則表現には合わないようだ．注 1.6.15 参照．
2. $L((0 + 1)^*) = \{0,1\}^*$．$+$ は和集合に対応する「または $\vee$」みたいなものである．（$a+$ と書いた時は $a + \epsilon$ と理解してもいい）．
3. $(0 + 1)^* 00 (0 + 1)^*$ は 2 つ以上続く 0 を持つ文字列全体の集合を表す．

1.6.1　正則表現の代数的構造

正則表現があからさまにするもの

$+, \cdot$ などの記法は，次の代数的法則が成り立つことによる．言語の代数的性質を救い上げている最高にクールな記法である．**連結を積とし，和集合を和とした代数系**のようになる．そして **Kleene star がまるで冪**である！なんと美しい．

補題 1.6.4 (代数的構造 : 集合代数から輸入)**.** 次の正則表現上の等式は，解釈写像 L と Set をモデルとして充たされる．(即ち，正則表現上の $=$ は解釈写像 L を用いて集合の相等として解釈する)．

1. $r + s = s + r$．和集合の可換性 $L(r) \cup L(s) = L(s) \cup L(r)$ より．
2. $(r + s) + t = r + (s + t)$．和集合演算の結合性から．
3. $(rs)t = r(st)$．連結の結合律．
4. $\emptyset + r = r + \emptyset = r$．空集合は $\cup$ の中立元．
5. $r(s + t) = rs + rt, (s + t)r = sr + tr$．分配則．連結してから和をとっても和をとってから連結しても可換であるため．
6. $r + r = r$．$\cup$ の吸収律より．
7. $(r^*)^* = r^*$．閉包の安定性．
8. $r^*r = rr^*$．

定義 1.6.5 (infinite regular expression)**.** 列 $(r_n)_{n\in\mathbb{N}}$ に対しても，

$$L((r_n)_{n\in\mathbb{N}}) := \bigcup_{n\in\mathbb{N}} L(r_n)$$

として解釈を与える．

記法 1.6.6. 正規表現 r^+ を rr^* の略記とする．$L(r^+) = L(r^*) \setminus \{\epsilon\}$ が成り立つ．

補題 1.6.7 (展開法則)**.** 正規表現 r, s について，次の式が成り立つ．

1. $r^* = \epsilon + r + rr + rrr + \cdots$．
2. $r^* = \epsilon + r^+$．
3. $(\epsilon + r)^* = r^*$．
4. $(r + s)^* = (r^*s^*)^*$．

1.6.2　正則表現の変数の一般化

正則表現を生成するアルファベットから抽象化し，独自の対象として定める

正則表現を形式科学的対象として離陸させるには，アルファベットを，言語の上を走る不定元からなる有限集合 $\Gamma := \{X_1, \cdots, X_m\}$ に定め直して，この上の正則表現を考える．
正則表現の各文字 r, s は，文字列の表象ではなく，言語の表象だと思ってもうまくいく．R を Γ 上の正則表現とすると，言語 $L(R) \subset \Gamma^*$ の元である各々の文字列に，言語を代入して全体を連結で解釈したもの $R(L_1, \cdots, R_m)$[a]と，Γ 上の正則表現としての文字列 R を置換 $R(L_1, \cdots, L_m)$ してから正則表現として解釈 $L(R(L_1, \cdots, L_m))$ するのとは，可換である（命題 1.6.9）．

$$\begin{array}{ccc} \mathrm{REGEX}_\Gamma & \xrightarrow[(L_1,\cdots,L_m)]{\text{代入}} & \mathrm{REGEX}_\Sigma \\ {\scriptstyle L}\downarrow & & \downarrow{\scriptstyle L} \\ P(\Gamma^*) & \xrightarrow[\cdot]{\text{連結}} & P(\Sigma^*) \end{array} \tag{1.1}$$

するとこの図式が可換であることから，正則表現という対象がついに離陸する．同値関係を 1.6.11 のように，「任意のアルファベットについて」と全称量化によって導入すれば，自立した安定な形式科学的道具である．まるでモデルを用いた一階述語論理の意味論の定義のように，正則表現も唯我独尊の対象となる．

[α] 文字列の代入と同じ記号を使っている

記法 1.6.8.

1. X がアルファベット Σ 上の**言語変数**とは，不定元 $X \in \mathsf{P}(\Sigma^*)$ とする．
2. この言語変数からなるアルファベット $\Gamma := \{X_1, \cdots, X_m\}$ 上の正則表現 R を考える．これを，使われ得る言語変数を明示して $R(X_1, \cdots, X_m)$ などと書く．この解釈は Γ 上の言語 $L(R) \subset \Gamma^*$ である（図式 1.1 の左下）．
3. また R は，言語変数に対するアルファベット Σ 上の言語の（文字列としての）代入 $(X_i) := (L_i)$ を定めるたびに，Γ 上の言語を定義する正則表現となり，これを $R(L_1, \cdots, L_m) := L(R(L_1, \cdots, L_m))$ と書く（図式 1.1 の右上周り）．

命題 1.6.9 (言語変数上の正則表現の解釈の特徴付け)**.** アルファベット $\Gamma = \{X_1, \cdots, X_m\}$ 上の正則表現 $R(X_1, \cdots, X_m)$ を考える．これに任意の言語 $L_1, \cdots, L_m \subset \Sigma^*$ を代入して得る正則言語 $R(L_1, \cdots, L_m) \subset \Sigma^*$ について，次が成り立つ：

$$\forall w \in R(L_1, \cdots, L_m),\ \exists X_{k_1} \cdots X_{k_m} \in L(R),\ w \in L_{k_1} \cdots L_{k_n}.$$

即ち，正則表現 R が定める正則言語 $R(L_1, \cdots, L_m)$ は，その解釈 $L(R) \subset \Gamma^*$ の解釈を言語の接合（定義 1.1.12）だと思ったものに等しい．しかし，言語変数の文字列の集合 $L(R) \subset \Gamma^*$ を適切に Σ 上の言語の代入に沿って Σ 上の正則言語と解釈する方法は簡単には思いつかないので，こうして命題として立て，理論の完成は定理に譲った．

［証明］. 正則表現の構成法に関する帰納法で証明する．任意に $L_1, \cdots, L_m \subset \Sigma^*$ を取る．

basis　1.　$R = \emptyset$ の時，$R(L_1, \cdots, L_m) = \emptyset$（かつ $L(R) = \emptyset$）なので，自明な形で定理は成立．

2.　$R = X_i$ の時，$L(R) = \{X_i\}, R(L_1, \cdots, L_m) = L(L_i) = L_i$ である．この時，任意の $w \in R(L_1, \cdots, L_m) = L_i$ について，$X_i \in L(R)$ が存在して，$w \in X_i(X_i := L_i)$[†7] $= L_i$ は自明に成立．

deduction　1.　$R = R_1 + R_2$ の時，$R(L_1, \cdots, L_m) = R_1(L_1, \cdots, L_m) \cup R_2(L_1, \cdots, L_m)$ であるため，成立．

2.　$R = R_1 \cdot R_2$ の時，$R(L_1, \cdots, L_m) = R_1(L_1, \cdots, L_m) \cdot R_2(L_1, \cdots, L_m)$ であるため，成立．

3.　$R = R_1^*$ の時，$R(L_1, \cdots, L_m) = R_1(L_1, \cdots, L_m)^*$ であるため，成立．

□

所感 1.6.10. こうして，図式 1.1 の左下から右下の写像が開通した．

定義 1.6.11 (言語変数上の正則表現の同値)**.** 言語変数 $\Gamma = \{X_1, \cdots, X_m\}$ 上の正則言語 R, S が同値であるとは，次を満たすことをいう：

$$R \equiv S :\Leftrightarrow \forall \Sigma,\ \forall L_1, \cdots, L_m \subset \Sigma^*,\ R(L_1, \cdots, L_m) = S(L_1, \cdots, L_m).$$

所感 1.6.12. まるでモデルのように，何のアルファベット代入しても同じ正則言語を生み出すものとして定義する．これが一体どのような効能を持っているかは恐ろしいが，結局 Γ^* 上の言語として一致することだとわかる．

定理 1.6.13 (言語変数上の正則表現の同値の特徴付け)**.** 言語変数 $\Gamma = \{X_1, \cdots, X_m\}$ 上の正則言語 R, S について，次の 2 条件は同値である．

1. $R \equiv S$.
2. $L(R) = L(S) \subset \Gamma^*$.

［証明］.

1⇒2　1 の定義は

$$R \equiv S :\Leftrightarrow \forall \Sigma,\ \forall L_1, \cdots, L_m \subset \Sigma^*,\ R(L_1, \cdots, L_m) = S(L_1, \cdots, L_m).$$

これに対して，$\Sigma = \Gamma$ とし，$L_i = \{X_i\}\ (i = 1, \cdots, m)$（これは確かに Γ 上の言語 $\{X_i\} \subset \Gamma^*$）を各変数 X_i に代入しても $R(L_1, \cdots, L_m) = S(L_1, \cdots, L_m)$ が成り立つはずである．しかしこれは解釈写像 L の R, S についての値に他ならない．従って，$L(R) = L(S)$.

[†7] 記号列に対する記号の代入の意味

2⇒1　アルファベット Σ を任意にとり，その上の言語 $L_1, \cdots, L_m \subset \Sigma^*$ を任意に定め，これについて

$$R(L_1, \cdots, L_m) = S(L_1, \cdots, L_m)$$

を示せば良い．命題 1.6.9 より，$w \in R(L_1, \cdots, L_m)$ ならば，ある言語変数の記号列 $X_{k_1} \cdots X_{k_n} \in L(R) \subset \Gamma^*$ が存在して，その解釈について $w \in L_{k_1} \cdots L_{k_n}$ となる．よって，$R(L_1, \cdots, L_m) \subset S(L_1, \cdots, L_m)$ 全く対称的な議論も成り立つから，$R(L_1, \cdots, L_m) = S(L_1, \cdots, L_m)$.

□

所感 1.6.14. まさに図式 1.1 が可換であることから示せる．

注 1.6.15 (正則表現に積演算 $\cap$ を入れない理由).

1. この定理はある意味で言語変数上の正則表現の well-definedness を表している．言語変数上の正則表現も，通常通りに解釈すれば，我々が自然に感じる「同値」と「正則表現としての同値」とが一致するように作れている．まるでモデル理論である．
2. 実は，正則言語は $\cap$ について閉じている（定理 1.8.2）が，正則表現に $\cap$ に相当する表現を組み込むと，定理の 2⇒1 が成り立たなくなる．例えば，正則表現に記号 $\cap$ を追加して，$R = X_1 \cap X_2 \cap X_3, S = X_1 \cap X_2$ ($\Gamma = \{X_1, X_2, X_3\}$) と定める．この時 $L(R) = L(S) = \emptyset \subset \Gamma^*$ であるが，$R(\{0\}, \{0\}, \emptyset) = \emptyset, S(\{0\}, \{0\}, \emptyset) = \{0\}$ となる．

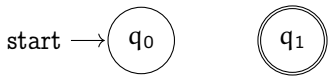
1.7　有限オートマトンと正則表現

前節で正則表現を定義したが，これが正確に正則言語を定める代数系であることを，２つの定理を立てて確認する．これで，正則表現が，有限オートマトンのどのような代数的構造を捉えていたかが明るみに出る．

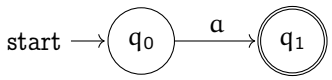
定理 1.7.1. 正則表現 ψ が定める言語 $L(\psi)$ は正則である．

［証明］. $L(\psi)$ を受理する ϵ 動作付き非決定性有限オートマトンを，正則表現 ψ の構成についての帰納法で構成する．なお，構成される ϵ-NFA は全て，初期状態に入る遷移はなく，また受理状態はただ一つであり，受理状態から出る遷移もないことに注意．

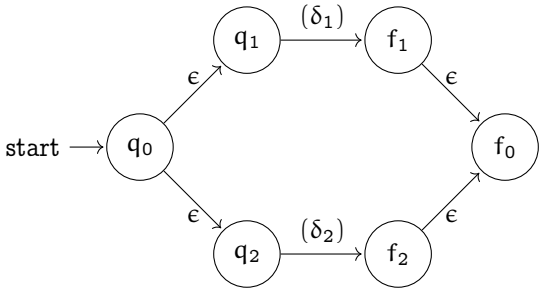
1. $\psi = \emptyset$ の時，次のオートマトンが $L(\psi) = \emptyset$ を受理する．



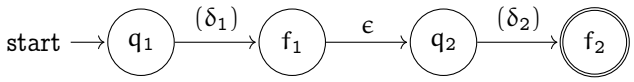
2. $\psi = a$ ($a \in \Sigma$) の時，次のオートマトンが $L(\psi) = \{a\}$ を受理する．



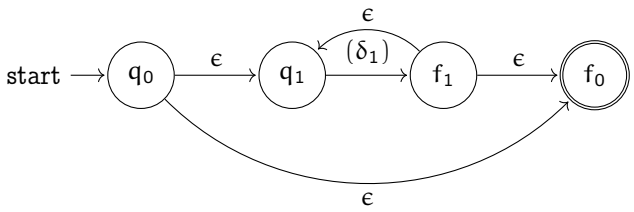
3. $\psi = r + s$ ($r, s \in \mathrm{REGEX}_\Sigma$) の時，$L(r), L(s)$ を受理する ϵ-NFA をそれぞれ $M_1 = (Q_1, \Sigma, \delta_1, q_1, \{f_1\}), M_2 = (Q_2, \Sigma, \delta_2, q_2, \{f_2\})$ とすると，M_1, M_2 は初期状態に入る遷移はなく，また受理状態はただ一つであり，受理状態から出る遷移もない．新たに $M_0 := (Q_1 \cup Q_2 \cup \{q_0, f_0\}, \Sigma, \delta_1 \cup \delta_2 \cup \{(q_0, \epsilon, q_1), (q_0, \epsilon, q_2), (f_1, \epsilon, f_0), (f_2, \epsilon, f_0)\}, q_0, \{f_0\})$ と定めれば良い（次図参照）．



4. $\psi = rs$ ($r, s \in \mathrm{REGEX}_\Sigma$) の時，$M_0 := (Q_1 \cup Q_2, \Sigma, \delta_1 \cup \delta_2 \cup \{(f_1, \epsilon, q_2)\}, q_1, \{f_2\})$ とすれば良い（次図参照）．



5. $\psi = r^*$ $(r \in \mathrm{REGEX}_\Sigma)$ の時，$M_0 := (Q_1 \cup \{q_0, f_0\}, \Sigma, \delta_1 \cup \{(q_0, \epsilon, q_1), (q_0, \epsilon, f_0), (f_1, \epsilon, q_1), (f_1, \epsilon, f_0)\}, q_0, \{f_0\})$ とすれば良い（次図参照）．



□

所感 1.7.2. 全ての ϵ-NFA について，初期状態に入る遷移はなく，また受理状態はただ一つであり，受理状態から出る遷移もない．

定理 1.7.3. 有限性決定オートマトン $M = (Q, \Sigma, \delta, q_1, F)$ が定める正則言語 $L(M)$ を表現する正則表現 ψ が存在する．

［証明］．$Q = \{q_i\}_{i \in [n]}$ とする．状態 q_i, q_j と整数 k の $1 \leqslant i, j, k \leqslant n$ を満たす 3-組について，次の 2 条件が成り立つ記号列 $x = a_1 \cdots a_m \in \Sigma^*$ を，記号列 x により q_i から q_j まで $\{q_1, \cdots, q_k\}$ に属する状態のみを経由して到達可能といい，そのような記号列 $x \in \Sigma^*$ 全体からなる集合を R_{ij}^k とする．

1. $\forall l \in \{1, \cdots, m-1\}$, $\delta^*(q_i, a_1 \cdots a_l) \in \{q_1, \cdots, q_k\}$.
2. $\delta^*(q_i, a_1 \cdots a_m) = \delta_j$.

いま，R_{ij}^{k-1} $(1 \leqslant i, j \leqslant n)$ が定義されているとき，新たに q_k を経由するものと，従来のものとに分けて，R_{ij}^k $(1 \leqslant k \leqslant n)$ は次のように帰納的に定義されている．

$$R_{ij}^0 = \begin{cases} \{a \in \Sigma \mid \delta(q_i, a) = q_j\}, & i \neq j \text{ のとき}, \\ \{a \in \Sigma \mid \delta(q_i, a) = q_j\} \cup \{\epsilon\}, & i = j \text{ のとき}. \end{cases}$$

$$R_{ij}^k = R_{ik}^{k-1}(R_{kk}^{k-1})^* R_{kj}^{k-1} \cup R_{ij}^{k-1} \qquad (\text{新たに } q_k \text{を経由するものとしないもの})$$

この構造に沿って，記号列の集合 R_{ij}^k を表現する正則表現 r_{ij}^k を次のように帰納的に定義する． □

所感 1.7.4. こんな構造見抜けるか？

1.8　正則言語の閉包属性

初め有限オートマトンという抽象機械が受理する言語として正則言語を定義し，3 つの抽象機械の等価性を見た．続いて，正則表現という等価な代数的手法も手に入れた．道具立ては揃った，続いて正則言語を調べていく．正則言語は，集合演算 $\cap, \cup$，接合 $\cdot$，補集合演算 $\_$，Kleene star $^*, ^+$，反転 ^{R}，差集合 $\setminus$，商集合 $/$，逆 C，準同型の像 $\mathrm{Im}\,\varphi$，準同型の逆像 $\mathrm{Ker}\,\varphi$，シャッフル $\vee$ について閉じている．極めて代数的で扱いやすい！！

記法 1.8.1. 正則言語からなる集合を $\mathfrak{R}$ とし，この性質を調べる．

1.8.1　代数的性質

定理 1.8.2 (代数的性質)**.** 正則言語からなる集合 $\mathfrak{R}$ は，集合演算 $\cap, \cup$，接合 $\cdot$，補集合演算 $\_$，Kleene star $^*, ^+$，反転 ^{R} の下で閉じている．

定理 1.8.3 (差集合)**.** 言語 $L_1, L_2 \subset \Sigma^*$ に対して，L_2 が正則ならば，次も正則である：

$$L_1 \setminus L_2 := \{y \mid \exists x \in L_1,\ xy \in L_2\}.$$

定理 1.8.4 (商集合)**.** 言語 $L_1, L_2 \subset \Sigma^*$ に対して，L_1 が正則ならば，次も正則である：

$$L_1/L_2 := \{x \mid \exists y \in L_2,\ xy \in L_1\}.$$

定理 1.8.5. $L \subset \Sigma^*$ は正則の時，次も正則である：

$$C(L) := \{xy \mid x, y \in \Sigma^*, yx \in L\}$$

1.8.2　射による正則言語の定義

言語の空間に定義されるモノイドの射 $h : \Sigma^* \to \Gamma^*$ によって，正則言語が写り合う構造が素晴らしい．

定義 1.8.6 (モノイドの射)**.** アルファベット Σ, Γ に対して，写像 $h : \Sigma^* \to \Gamma^*$ が次の条件を満たす時，**準同型**という：

1. $h(\epsilon) = \epsilon$.
2. $\forall u, v \in \Sigma^*,\ h(uv) = h(u)h(v)$.

命題 1.8.7 (モノイドの基底)**.** $f, g : \Sigma^* \to \Gamma^*$ を準同型とする．$(\forall a \in \Sigma,\ f(a) = g(a)) \to f = g$.

定理 1.8.8. Σ, Γ をアルファベットとし，$L \subset \Sigma^*, L' \subset \Gamma^*$ を正則言語とする．準同型 $h : \Sigma^* \to \Gamma^*$ に対して，次が成り立つ：

1. 像 $h(L)$ は正則.
2. 逆像 $h^{-1}(L')$ は正則.

命題 1.8.9 (shuffle)**.** 正則集合 $L_1, L_2 \subset \Sigma^*$ に対して，次の集合も正則：

$$L_1 \vee L_2 = \{u_1 v_1 \cdots u_n v_n \mid u_1, \cdots, u_n \in L_1, v_1, \cdots, v_n \in L_2, u_i, v_i \in \Sigma^*\}.$$

1.9　ブール行列による有限オートマトンの表現

算譜に特性関数のアイデアが使えるので，NFA は Bool 行列のデータ構造に翻訳できる

再び正則表現とは別の代数的表現を思いついた！有限オートマトンを計算機上にコードし模倣するための準備として，ブール行列での表現を考える．
確かに正則言語の構造は，自由モノイドの上に建設されて極めて代数的であった．環の射っぽくもあった．
こうしてさらに，正則言語 L は写像 L'，$\sqrt{L}$ について閉じていることがわかった．

定義 1.9.1 (Boolean algebra)**.** 次のように定義された代数系 $(2, +, \cdot)$ を**ブール代数**という．

$$0 + 0 = 0, \qquad 1 + 0 = 0 + 1 = 1 + 1 = 1,$$
$$0 \cdot 0 = 0 \cdot 1 = 1 \cdot 0 = 0, \qquad 1 \cdot 1 = 1.$$

命題 1.9.2. $(2, +, \cdot)$ をブール代数とする．

1. $+$ について，吸収的な加法群である．
   (1) $x + 0 = 0 + x = x$.
   (2) $x + (y + z) = (x + y) + z$.
   (3) $x + y = y + x$.
   (4) $x + x = x$.
2. $\cdot$ についても，吸収的な加法群である．
   (5) $1 \cdot x = x \cdot 1 = x$.
   (6) $x \cdot (y \cdot z) = (x \cdot y) \cdot z$.
   (7) $x \cdot y = y \cdot x$.

(8) $x \cdot x = x$.

3. 2 つの演算について，分配則が成り立つ．

(9) $x \cdot (y + z) = x \cdot y + x \cdot z$.

1.9.1　ブール行列

NFA とは，$\Delta : \Sigma \to \mathrm{Map}(Q \times Q, 2)$ で表せる．ある文字 $a \in \Sigma$ に対して，$\Delta^a := ((p, a, q) \in \delta)_{p,q \in Q}$ の真理値を集めた行列として表現できる．これはグラフ理論でも用いられる手法だろう．平面的で圏的ならば，二方向的なので，行列としての二次元表現を持つ．

定義 1.9.3 (Boolean matrix)**.** 有限の状態集合 Q に対して，$Q \times Q$ **ブール行列** M とは，写像 $M : Q \times Q \to 2$ のことである．行列積は，Bool 代数の和と積について行う．$1 \times Q$ 行列のことを**ブールベクトル**という．

定義 1.9.4 (Bool 行列による NFA の表現 Δ)**.** 非決定性有限オートマトン $M = (Q, \Sigma, \delta, q_0, F)$ に Bool 行列の集合を対応させる写像

$$\begin{array}{ccc} \Delta : \Sigma & \longrightarrow & \mathrm{Map}(Q \times Q, 2) \\ \cup & & \cup \\ a & \longmapsto & \Delta_a \end{array}$$

を，Δ_a を δ の特性関数 $\Delta_a(p, q) = 1 :\Leftrightarrow (p, a, q) \in \delta$ とする．

Δ の指数表記を

1. $x = a_1 \cdots a_n$ の時，$\Delta^x = \Delta^{a_1} \cdots \Delta^{a_n}$ とし，
2. $\Delta^\epsilon = E$（単位行列）

とする．

命題 1.9.5. 記号列 $x = a_1 \cdots a_n$ と状態 $p, q \in Q$ について，次の 2 条件は同値である．

1. $\Delta^x(p, q) = 1$.
2. M に於ける状態遷移の列 $p \xrightarrow{a_1} p_1 \cdots p_{n-1} \xrightarrow{a_n} p_n$ が存在する．

系 1.9.6. $x \in \Sigma^*$ について，次の 2 条件は同値である．

1. $\displaystyle\sum_{q \in F} \Delta^x(q_0, q) = 1$.
2. x は M によって受理される．

1.9.2　Bool 行列の応用

NFA の受理を，行列とその積のデータ構造に落とし込めた．こんな途方もないデータ構造を作ると，見えてくる証明がある．

記法 1.9.7. 状態の部分集合 $S \subset Q$ に対して，Q ブール列ベクトル I_S を特性関数 $I_S(p) = 1 :\Leftrightarrow p \in S$ で定める．また，$p \in Q$ については $I_{\{p\}}$ を I_p と略記する．

系 1.9.8. 非決定性有限オートマトン $M = (Q, \Sigma, \delta, q_0, F)$ について，

$$L(M) = \{x \in \Sigma^* \mid I_{q_0} \Delta^x {}^t I_F = 1\}$$

［証明］．$I_{q_0} \Delta^x {}^t I_F = 1$ は，$\Delta^x {}^t I_F$ の部分で，各 $q \in Q$ から x を読み込むことで終了状態への遷移が可能かを格納した縦 Bool ベクトルを得る．これと I_{q_0} はある一つの要素のみが 1 であるベクトルであるが，この filter を潜り抜けて 1 が通るかどうかが判定条件となり，これは受理計算が存在することに同値． □

定理 1.9.9. $L \subset \Sigma^*$ を正則言語とする．次のように定義される言語 L' も正則である．

$$L' := \{x \in \Sigma^* \mid |x| = |y| \text{ である } y \in \Sigma^* \text{が存在して } xy \in L\}.$$

［証明］. L を受理する DFA $M = (Q, \Sigma, \delta, q_0, F)$ に対して，$\mathcal{M}(Q) := \mathrm{Map}(Q \times Q, 2)$ を $Q \times Q$-Bool 行列全体からなる集合とする．次のような DFA $M' := (Q', \Sigma, \delta', q_0', F')$ を考える．

1. $Q' := Q \times \mathcal{M}(Q)$．現在状態と，$Q$ 上のあり得る Σ 上の NFA δ の組全体からなる集合．
2. $q_0' := (q_0, I)$．M の開始状態と，Q 上の離散的な $\emptyset$ を受理言語とする NFA の組．
3. $\delta'((p, A), a) := (\delta(p, a), A\Delta)$．ただし，$\Delta := \sum_{b \in \Sigma} \Delta^b$ を論理和とする．$A\Delta$ は，Bool 行列 A で定義される DFA について，遷移が存在するかと遷移できるかで論理積をとっていることに当たる．
4. $F' := \{(p, A) \in Q \times \mathcal{M}(Q) \mid I_p A^t I_F = 1\}$．現在状態を $p \in Q$ として，A が定める Q 上の遷移のあり方で，受理状態までの受理計算があるかを判定している．

さて，この DFA M' に対して $x = a_1 \cdots a_n \in L(M')$ とは，

$$\begin{aligned}
\delta'^*(q_0', x) &= \delta'^*((q, I), a_1 \cdots a_n) \\
&= \delta'^*(\delta'((q_0, I), a_1), a_2 \cdots a_n) && \text{一文字受理した} \\
&= \delta'^*((\delta(q_0, a_1), I\Delta), a_2 \cdots a_n) && \text{ルール 3 より} \\
&= \delta'^*((\delta^*(q_0, a_1 a_2), I\Delta\Delta), a_3 \cdots a_n) && \text{同じ} \\
&\ \ \vdots \\
&= (\delta^*(q_0, a_1 \cdots a_n), \Delta^n) \in F'
\end{aligned}$$

即ち，$I_{\delta^*(q_0, x)} \Delta^n {}^t I_F = 1$ に同値であるが，これは DFA M において，状態 $\delta^*(q_0, x) \in Q$ から長さ n の Δ^n で表される或る記号列 y によって[†8]，M の受理状態に到達できることを表している：$xy \in L$．従って，$L(M') = L'$． □

定理 1.9.10. $L \subset \Sigma^*$ を正則言語とする．次のように定義される言語 $\sqrt{L}$ も正則である．

$$\sqrt{L} := \{x \in \Sigma^* \mid xx \in L\}.$$

1.10　最小オートマトン

有限オートマトンのクラスの中で，状態数が一番少ないものを構成する算譜を M_L, M_L', M^μ の 3 通りで与える．

1.10.1　Myhill-Nerode の定理

定義 1.10.1 (index)**.** 同値類の濃度をその**指標**という．

定義 1.10.2 (right-invariant)**.** 集合 Σ^* 上の同値関係 R が次を満たす時，**右不変**であるという：

$$xRy \Rightarrow (\forall z \in \Sigma^*,\ xzRyz).$$

定義 1.10.3. 言語 $L \subset \Sigma^*$ が定める同値関係 R_L を

$$xR_Ly :\Leftrightarrow (\forall z \in \Sigma^*\ xz \in R_L \leftrightarrow yz \in R_L)$$

とし，$x \in \Sigma^*$ の同値類を $[x]_L$ で表す．

命題 1.10.4.

1. R_L は確かに同値関係である．

[†8] Δ^n のデコードができないが？ y は同じ文字の n 連続ってことか？

2. R_L は右不変である：
$$[x]_L = [y]_L \Rightarrow (\forall z \in \Sigma^* \ [xz]_L = [yz]_L).$$

命題 1.10.5. 言語 L に対して，次のように構成される決定性オートマトンは L を受理する．

1. $Q_L = \{[x]_L \mid x \in \Sigma^*\}$.
2. $\delta_L([x]_L, a) = [xa]_L$ $(x \in \Sigma^*, a \in \Sigma)$.
3. $q_0^L = [\epsilon]_L$.
4. $F_L = \{[x]_L \mid x \in L\}$.

定理 1.10.6 (Myhill-Nerode)**.** 言語 $L \subset \Sigma^*$ について，次の 3 条件は同値である．

1. L は正則．
2. 有限指標で右不変な同値関係 R が存在して，L はその同値類に直和分割される．
3. R_L は有限指標である．

定理 1.10.7. 言語 $L \subset \Sigma^*$ が命題 1.10.5 で定める決定性オートマトンを $M_L = (Q_L, \Sigma, \delta_L, q_0^L, F_L)$ とする．決定性オートマトン $M = (Q, \Sigma, \delta, q_0, F)$ も L を受理するならば，全射 $\varphi : Q \to Q_L$ が存在する．

系 1.10.8. M_L は L を受理する状態数が最小の決定性オートマトンである．

1.10.2　決定性オートマトンの圏

命題 1.10.9. 決定性オートマトン $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ が $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ を模倣するとする．$L(M_1) = L(M_2)$ である．

定義 1.10.10 (isomorphism)**.** 決定性オートマトンの射 $\varphi : M_1 \to M_2$ が，全単射で，1,2 の代わりに次の 2 条件を満たす時，**同型**という．

1. $\varphi(q_1) = q_2$.
2. $\varphi(F_1) = F_2$.

定理 1.10.11 (最小オートマトンの同型を除いた一意性)**.** 正則言語 L について，L を受理する状態数が最小の有限性決定オートマトンは同型を除いて一意に定まる．

1.10.3　別構成

定義 1.10.12. 言語 $L \subset \Sigma^*$ と $x \in \Sigma^*$ に対して，言語 $x \setminus L$ を次のように定める：
$$x \setminus L := \{y \mid xy \in L\}.$$

命題 1.10.13. 言語 $L \subset \Sigma^*$ について，次の 2 条件は同値である．

1. $xR_L y$.
2. $x \setminus L = y \setminus L$.

定理 1.10.14. 言語 L が定める最小オートマトンを $M_L = (Q_L, \Sigma, \delta_L, q_0^L, F_L)$（命題 1.10.5）とする．これは次のオートマトン $M'_L = (Q'_L, \Sigma, \delta'_L, q_0'^L, F'_L)$ と同型である．

1. $Q'_L = \{x \setminus L \mid x \in \Sigma^*\}$.
2. $\delta'_L(X, a) = a \setminus X$ $(a \in \Sigma, X \in Q'_L)$.
3. $q_0'^L = L$.
4. $F'_L = \{X \in Q'_L \mid \epsilon \in X\}$.

1.10.4　状態数の最小化

状態に同値関係を導入して，割ることを考える．

定義 1.10.15 (equivalent, distinguishable)**.** $M = (Q, \Sigma, \delta, q_0, F)$ を DFA とする．

1. $p, q \in Q$ が**同値**であるとは，$p \equiv_M q :\Leftrightarrow \forall z \in \Sigma^*\ \delta^*(p, z) \in F \leftrightarrow \delta^*(q, z) \in F$.
2. $p \not\equiv_M q$ であることを，**区別可能**という．

定理 1.10.16. $M = (Q, \Sigma, \delta, q_0, F)$ を全ての状態が初期状態から到達可能な DFA とする．この時，次のように構成される DFA M^μ は M_L と同型である．

1. $Q^\mu = \{[q]_{\equiv_M} \mid q \in Q\}$.
2. $\delta^\mu([q]_{\equiv_M}, a) = [\delta(q, a)]_{\equiv_M}$ $([q]_{\equiv_M} \in Q^\mu, a \in \Sigma)$.
3. $q_0^\mu = [q_0]_{\equiv_M}$.
4. $F^\mu = \{[q]_{\equiv_M} \mid q \in F\}$.

命題 1.10.17. $M = (Q, \Sigma, \delta, q_0, F)$ を DFA とする．任意の $p \in F$ と任意の $q \in Q \setminus F$ に対して，p, q は区別可能である．

1.11　その他の有限状態機械

オートマトンの変遷

有限オートマトンの拡張として，様々な有限状態機械を考える．
DFA では退化していた出力の概念を分離して取り出す．出力アルファベット Δ と出力関数 $\gamma : Q \times \Sigma \to \Delta$ の追加は等価な抽象機械を定義し，これを順序機械（過去の入力を記憶しているという意味で）という．変換器 (transducer) ともいう．これには出力の仕方について 2 種類存在し，

1. 状態と出力が対応するものを Moore 機械 [5],
2. 遷移（＝状態と入力の組）と出力が対応するものを Mealy 機械 [4]

という．Moore は Mealy が退化した形とみなせる．論理回路（フリップフロップなど）が順序機械の例である．機能としては全く等価である（還元算譜が存在する）．

1.11.1　順序機械

transducer の応用

記号列を記号列に出力する問題 $\Sigma^* \to \Sigma^*$ への算譜となるのが順序機械である．

- 音声認識，音声合成：音＝音素列 $\to$ 単語列．
- かな漢字変換，読み推定：かな漢字列 $\leftrightarrow$ かな列．
- 形態素解析：文字列 $\to$ 単語列．
- 機械翻訳：単語列 $\leftrightarrow$ 単語列．
- 固有表現認識：単語列 $\to$ 固有表現列．
- 文法誤り訂正：単語列 $\to$ 単語列．

例 1.11.1. vending machine はとても良い有限状態機械の例である．

定義 1.11.2 (sequential machine, transducer (Mealy))**.** 次の 5 条件を満たす 6-組 $S = (Q, \Sigma, \Delta, \delta, \gamma, q_0)$ を**順序機械**または**変換**

器という．

1. Q は有限集合で，$q_0 \in Q$ は初期状態．
2. Σ は入力アルファベット．
3. Δ は出力アルファベット．
4. $\delta : Q \times \Sigma \to Q$ は状態遷移関数．
5. $\gamma : Q \times \Sigma \to \Delta$ は出力関数．

$(q, a) \in Q \times \Sigma$ に対して $\delta(q, a) = p, \gamma(q, a) = b$ であることを，$q \xrightarrow{a/b} p$ と書く．

定義 1.11.3 (順序機械が計算する関数)**.** 順序機械 S が関数 $r_S : \Sigma^* \to \Delta^*$ を**計算する**とは，入力 $a_1 \cdots a_n \in \Sigma^*$ に対して，

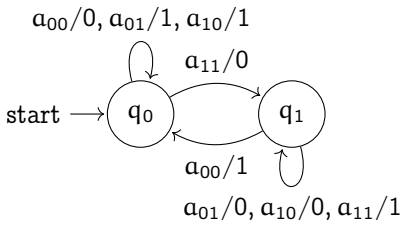
$$q_0 \xrightarrow{a_1/b_1} q_1 \cdots q_{n-1} \xrightarrow{a_n/b_n} q_n$$

となる時，記号列 $b_1 \cdots b_n \in \Delta^*$ を**出力**といい，$r_S(a_1 \cdots a_n) = b_1 \cdots b_n$ であることとする．なお，$r_S(\epsilon) = \epsilon$ とする．

例 1.11.4 (加算器：Mealy 機械では carry に依って状態は 2 つ用意すれば十分)**.**

$$\Sigma = \{a_{00}, a_{01}, a_{10}, a_{11}\}, \qquad \Delta = \{0, 1\}.$$

とし，2 つの二進数 $x_{n-1} \cdots x_1 x_0, y_{n-1} \cdots y_1 y_0 \in \{0, 1\}^*$ を加える論理回路を設計したい．記号列 $a_{x_0 y_0} a_{x_1 y_1} \cdots a_{x_{n-1} y_{n-1}}$ を次の順序機械に入力すれば良い．なお，出力を $b_n b_{n-1} \cdots b_1 b_0$ とすれば，最終状態が q_0 の時繰り上がりはないので $b_n = 0$，最終状態が q_1 の時 $b_n = 1$ と γ を定める．



命題 1.11.5 (Moore 機械と Mealy 機械の等価性)**.** 出力関数が $\gamma : Q \to \Delta$ に退化した，即ち各状態 q_1 に対して出力 $q_1/1$ が決まっている機械を Mealy 機械を Moore 機械という．Moore 記号の方が記号を多く出すが，実質的には等価である．

［証明］.

Moore⇒Mealy　遷移先の状態に対応した出力を，遷移に対して定義する．即ち，Moore 機械の出力関数を $\gamma : Q \to \Delta$ とし，遷移関数を $\delta : Q \times \Sigma \to Q$ すると，Mealy 機械の出力関数 $\gamma' : Q \times \Sigma \to \Delta$ は，$\gamma'(q_1, a) = \gamma(\delta(q_1, a))$ とすれば良い．

Mealy⇒Moore　出力文字 Δ に応じて，状態数をふやす．Mealy 機械を $M = (Q, \Sigma, \Delta, \delta, \gamma, q_0)$ とすると，新たに $Q' = \{q/a\}_{q \in Q, a \in \Delta}$ とする．遷移 δ', γ' をこれに従って定めると良い．

□

1.11.2　一般化順序機械が定める関数とモノイドの射

定義 1.11.6 (generalized sequential machine)**.** 順序機械 $S = (Q, \Sigma, \Delta, \delta, \gamma, q_0)$ の出力関数の値が文字ではなく文字列である時，出力関数は $\gamma : Q \times \Sigma \to \Delta^*$，これを**一般化順序機械**という．

定理 1.11.7 (一般化順序機械が計算する関数は正則構造を保つ)**.** Σ, Δ をアルファベットとする．$L \subset \Sigma^*$ 及び $L' \subset \Delta^*$ を正則言語とする．一般化順序機械 $S = (Q, \Sigma, \Delta, \delta, \gamma, q_0)$ が計算する関数 $r_S : \Sigma^* \to \Delta^*$ について，次が成り立つ．

1. $r_S(L)$ は正則．
2. $r_S^{-1}(L')$ は正則．

命題 1.11.8. モノイド準同型 $h : \Sigma^* \to \Delta^*$ は一般化順序機械によって計算できる．

1.11.3　隠れマルコフモデル

確率モデルと有限オートマトンを組み合わせたようなもの.[†9]状態遷移に，確率と考えられる重みをつけたもの．そしてスコアが最大になるものを探す.

1.11.4　ω-オートマトン

可算濃度の ω である．無限長の文字列を受理するオートマトンで，Büchi automaton や Rabin automaton などがあり，動き続けるシステムのモデルとなる．特に形式手法・形式モデルの分野でシステムのモデルとして考えられる.

1.12　有限オートマトンについてのアルゴリズム

[†9] 3A 知能システム論.

第 2 章

文脈自由文法とプッシュダウンオートマトンの理論

一段階議論の抽象度をあげる．

1. 文脈自由言語を，文法が生成するものとして定義し，文法を研究対象とする．文法が対象言語で，これに対してメタ言語で生成関係 $\Rightarrow$ を定義して研究する．V が状態の集合，T がアルファベット，P が遷移規則の一般化と見れる（遷移規則は正則文法として相対化される）．
2. 文法 G の標準形を目指して，3 段階に分けて簡素化のアイデアを考える．
3. 空語を省いた標準形である **Chomsky 標準形**に到達する．これは特に生成規則を理解しやすい標準形である．実際，構文木が二分木になる為，二分木化 (binarization) と呼ばれる．
4. 文脈自由言語には，鳩の巣原理により生じる大域的な繰り返し模様がある．これを文脈自由言語の必要条件として捉えたのが Ogden の補題，またはその特殊化である反復補題である．
5. 所属問題は構文木の存在問題である．chart というデータ構造の上に構文木の構成を試みる手法は些か Jordan 標準形に似ている．コンパイラでは使われない計算量だが，文脈自由言語全般につけるため自然言語処理でも使える動的計画法である CYK アルゴリズムを与える．
6. 正則言語の文脈自由言語のクラスとしての特徴付け：正則文法を与え，文脈自由言語の真のクラスであることを見る．
7. 文脈自由言語のクラスにふさわしい抽象機械が，プッシュダウンオートマトンである．これを与えるのが Greibach 標準形の理論である．
8. 以降の計算可能な言語の階層分けの世界観として，Chomsky 階層がある．

2.1 定義と例

文脈自由文法 G は，生成する記号列 T^* を定める．これを文脈自由言語といい，生成途中の $T^* \setminus (V \cup T)^*$ の元を文形式という．正則言語論でいうアルファベット Σ とは，終端記号 T として相対化されたことになる．

定義 2.1.1 (context-free grammar, variable / non-terminal, terminal (symbol), production, start symbol)**．文脈自由言語**とは，次の条件を満たす 4 つ組 $G = (V, T, P, S)$ のことをいう．

- V は**変数（非終端記号）**からなる有限集合，
- T は**終端記号**からなる有限集合で，V と交わらない，
- P は**生成規則**

$$\begin{array}{ccc} V & \longrightarrow & (V \cup T)^* \\ \cup & & \cup \\ A & \longmapsto & \alpha \end{array}$$

からなる有限集合

- $S \in V$ は**開始記号**．

定義 2.1.2 (derivation)**．**文脈自由文法 $G = (V, T, P, S)$ について，記号列を $\alpha_1, \cdots, \alpha_m \in (V \cup T)^*$ とする．

1. **直接的に導出される**とは，次の二項関係 $\Rightarrow_G$ をいう：

$$\alpha_1 \Rightarrow_G \alpha_2 \quad :\Leftrightarrow \quad \exists A \in V,\ \exists \alpha \in (V \cup T)^*,\ [((A, \alpha) \in P) \wedge (\exists \beta_1, \gamma_1 \in (V \cup T)^*,\ \alpha_1 = \beta_1 A \gamma_1, \alpha_2 = \beta_1 \alpha \gamma_1)]$$

2. **導出される**とは，二項関係 $\Rightarrow_G$ の反射的推移閉包 $\Rightarrow_G^*$ のことをいい，この時の $\Rightarrow_G$ の列のことを**導出**という．
3. どの文脈自由文法 G の下で議論しているかが明白である場合，下付き文字を落として $\Rightarrow, \Rightarrow^*$ などと表す．

定義 2.1.3 (context-free language, sentential form)**.** 文脈自由文法 G が**生成する言語** $L(G)$ とは，記号列の集合

$$L(G) := \{w \in T^* \mid S \Rightarrow_G^* w\}$$

のことをいう．

- 関係 $w \in L(G)$ を G は w を**生成する**という．
- このようにして文脈自由文法が定める言語を**文脈自由言語**という．
- 記号列 $\alpha \in (V \cup T)^*$ であって，$S \Rightarrow_G^* \alpha$ を満たすものは，**文形式**という．

例 2.1.4. $G = (V, T, P, S)$ を次のように定める：

- $V = \{S\}$,
- $T = \{a, b\}$,
- $P = \{(S, aSb), (S, ab)\}$.

が生成する文脈自由言語は，$L(G) = \{a^n b^n \mid n \in \mathbb{N} \setminus \{0\}\}$.

例 2.1.5. $G = (V, T, P, S)$ を次のように定める：

- $V = \{S, A\}$,
- $T = \{a, b\}$,
- $P = \left\{ \begin{array}{l} (S, aB), (S, bA), \\ (A, a), (A, aS), (A, bAA), \\ (B, b), (B, bS), (B, aBB) \end{array} \right\}$.

これが定める文脈自由文法は $L(G) = \{w \in \{a, b\}^+ \mid w$ に現れる a, b の数は等しい $\}$ である．$\subset$ はすぐに従う．$\supset$ も $ab, ba \in L(G)$ から帰納的にわかる．

例 2.1.6 (正則表現)**.** アルファベット $\{a, b\}$ 上の正則表現とは，次の文脈自由言語である：

- $V = \{E\}$,
- $T = \{a, b, (,), *, \cdot, +, \emptyset\}$,
- $P = \{E \to a \mid b \mid \emptyset \mid (E + E) \mid (E \cdot E) \mid (E^*)\}$.

例 2.1.7.

1. $S \to aSa \mid bSb \mid aa \mid bb$ は $\{ww^R \mid w \in \{a, b\}^+\}$ を生成する．
2. $S \to aB \mid bbA,\ A \to a \mid aS \mid bbAA,\ B \to bb \mid bbS \mid aBB$ は $\{w \in \{a, b\}^+ \mid w$ に出現する b の個数は a の 2 倍 $\}$ を生成する．

命題 2.1.8 (文脈自由言語の閉包属性)**.** 文脈自由言語 L_1, L_2 に対して，次の集合 L も文脈自由言語である．即ち，$\exists G$: 文脈自由文法, $L = L(G)$.

1. $L_1 \cup L_2$
2. $L_1 \cdot L_2$
3. L_1^*
4. L_1^R

［**証明**］. L_1, L_2 を生成する文脈自由文法をそれぞれ $G_1 = (V_1, T_1, P_1, S_1), G_2 = (V_2, T_2, P_2, S_2)$ とする．$V_1 \cap V_2 = \emptyset$ としても一

般性は失われない．$G = (V, T, P, S)$ を次のように定めれば良い．まず，$S \notin V_1, S \notin V_2$ とし，$V = V_1 \cup V_2 \cup \{S\}$，$T = T_1 \cup T_2$ とする．

1. $P = P_1 \cup P_2 \cup \{(S, S_1), (S, S_2)\}$ とすれば良い．
2. $P = P_1 \cup P_2 \cup \{(S, S_1S_2)\}$ とすれば良い．
3. $P = P_1 \cup P_2 \cup \{(S, S_1S_1)\}$ とすれば良い．
4. 次の写像の値域を P とすれば良い：

$$\begin{array}{ccc} P_1 \cup P_2 & \longrightarrow & V \times (V \cup T)^* \\ \cup\!\!| & & \cup\!\!| \\ (A, \alpha) & \longmapsto & (A, \alpha^R). \end{array}$$

□

命題 2.1.9. $L_1 \cap L_2$ は文脈自由言語とは限らない．

［証明］．$V = \{S, L, R\}, T = \{a, b, c\}$ とする．

$$P_1 := \left\{ \begin{array}{l} S \to LR, \\ L \to abL \mid ab, \\ R \to Rc \mid c \end{array} \right\}, \qquad P_2 := \left\{ \begin{array}{l} S \to LR, \\ L \to aL \mid a, \\ R \to Rbc \mid bc \end{array} \right\},$$

と定める．文脈自由文法を $G_1 := (V, T, P_1, S), G_2 := (V, T, P_2, S)$ とし，文脈自由言語を $L_1 := L(G_1), L_2 := L(G_2)$ とすると，

$$L_1 = \{a^n b^n c^m \mid n, m \geqslant 1\}, \qquad L_2 = \{a^n b^m c^m \mid n, m \geqslant 1\},$$

となる．$L := L_1 \cap L_2 = \{a^n b^n c^n \mid n \geqslant 1\}$ を考えると，これは文脈自由言語ではない（系 2.4.4）． □

文脈自由文法は言語を定義するための形式的手段だと理解した．そして手段であるだけでなく，形式科学的対象でもあるところが，極めて素性が良い研究対象となる所以なのであろう．

2.1.1　導出の木構造

直接導出関係 $\Rightarrow_G$ の推移閉包は，木構造を持つ．先生は，弁慶読みの例をあげ，意味論が宿るのは木構造分解においてだと論ずる．

> 文は，それが文法からどのように導出されたかによって意味をもつ．そこで，その導出をわかりやすく表すために木構造で表現することを考える．

そこで，構文木の存在性で捉える．分解の順番は関与しないが，本質的な差には関与するので，存在しえる構文木の数で意味の**曖昧さ**を定義する．

定義 2.1.10 (derivation tree / parse tree)**.** 文脈自由文法 $G = (V, T, P, S)$ が定める**導出木**または**構文木**とは，次を満たす木をいう：

1. $V \cup T \cup \{\epsilon\}$ でラベル付された節をもち，
2. ただし内部ノード（葉でないノード）のラベルは V の元で，
3. S を根とし，
4. 内部ノード v のラベル A とその子 $v_1, \cdots, v_k$ のラベルの列 $X_1 \cdots X_k$ に対して $A \to X_1 \cdots X_k \in P$ で，
5. ノード v のラベルが ϵ である場合は，ノード v が葉で，v は親ノードの唯一の子である場合に限る．

導出木の葉のラベルを左からつなげて $\alpha \in (V \cup T)^*$ となる場合，これを α を生む (yield) G-構文木という．

定理 2.1.11 (導出関係の木構造による定式化)**.** 文脈自由文法 $G = (V, T, P, S)$ について，次の 2 条件は同値．

1. $S \to^* \alpha$

2. α を生む G の構文木が存在する

［証明］. $S \to^* \alpha$ と構文木を一対一対応させる帰納法による. □

所感 2.1.12. 木構造は導出経路を捉えないので，この定式化を一番安定的に提供するデータ構造となっている．木構造は射 $\alpha_1 \to \alpha_m$ に対応するのであって，例えば，最右導出と最左導出などの分解の仕方（特に順番）には関与しないことが多い．これを使って概念を定義する．

定義 2.1.13 (leftmost derivation, rightmost derivation)**.** 導出 $\alpha_1 \Rightarrow \cdots \Rightarrow \alpha_m$ であって，各 $\alpha_i \Rightarrow \alpha_{i+1}$ $(i = 1, \cdots, m-1)$ について一番左端の変数が変化している時，これを**最左導出**という．同様に**最右導出**も定義する．

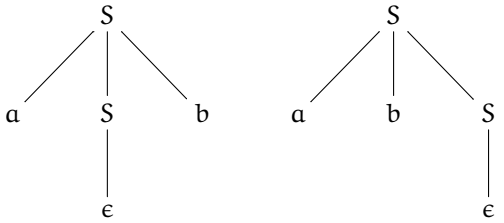
定義 2.1.14 (ambiguous, inherently ambiguous)**.**

1. 文脈自由文法 $G = (V, T, P, S)$ について，記号列 $w \in T^*$ が存在して，これを生み出す異なる G-構文木が 2 つ以上存在する時，G は**曖昧**であるという．
2. 文脈自由言語 L について，これを生成する文脈自由文法 G が常に曖昧である時，L は**本質的に曖昧**であるという．

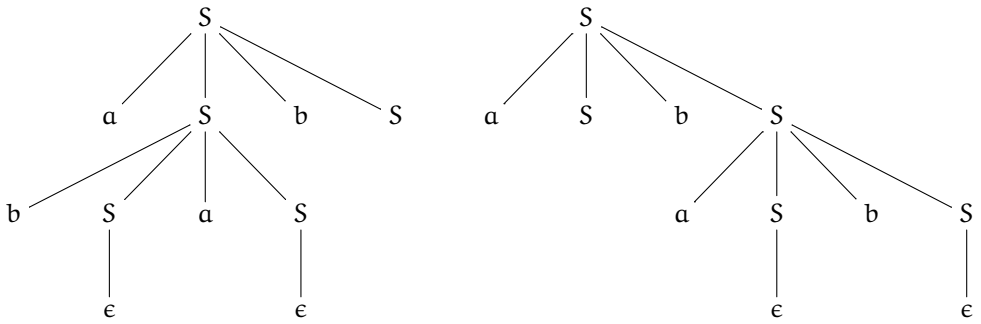
例 2.1.15. 次の文脈自由文法 $V = \{S\}, T = \{a, b\}$ は曖昧である．

1. $S \to aSb \mid abS \mid \epsilon$.
2. $S \to aSbS \mid bSaS \mid \epsilon$.

$ab \in T^*$ について，次の 2 つの構文木が存在するからである：



$abab \in T^*$ について，次の 2 つの構文木が存在するからである：



定理 2.1.16. 言語

$$L = \{a^n b^n c^m d^m \mid n, m \geqslant 1\} \cup \{a^n b^m c^m d^n \mid n, m \geqslant 1\}$$

は本質的に曖昧である．

2.2　文法の簡素化

無駄な記号がなく，空語生成規則と単一生成規則を持たない文脈自由文法

複雑なデータ構造を定義したときはいつでもそうだが，有限オートマトンに無駄な状態や遷移があったように，文脈自由文法にも無駄な変数や生成規則があったりする．これを剪定してある種の緩い標準形を得ることを考える．この標準形を 3 段階に分けて考えて，最後に無駄な記号がなく，空語生成規則と単一生成規則を持たない文脈自由文法を得る（定理 2.2.8）こととする．

ただしオートマトンと違うのはデータ構造の次元である．今回の対象となるデータは「文法規則」である．この次元の違い

は生成規則 $\to$ と二項関係 $\Rightarrow$ との使い分けにも現れている．

2.2.1　無駄な変数の除去

状態についての**到達可能性**（定義 1.2.20）と同じように，記号 X も，G-導出が存在してそれを踏むかどうかで有用性を判定する．圏論的な世界観の下で，全く共通した手法である．
この除去算譜は 2 つに分解して考える．補題を 2 つ立てる．

定義 2.2.1 (usefulness)**.** 記号 $X \in V \cup T$ について，ある $\alpha, \beta \in (V \cup T)^*$ が存在して，G-導出 $S \Rightarrow^* \alpha X \beta \Rightarrow^* w$ が存在するとき，**有用**であるという．有用でない記号を**無駄な記号**という．

補題 2.2.2 (終端記号を導出しない変数の除去算譜)**.** $G = (V, T, P, S)$ を $L(G) \neq \emptyset$ を満たす文脈自由文法とする．次の 2 条件を満たす文脈自由文法 $G' = (V', T, P', S)$ を構成する算譜が存在する．

1. $L(G) = L(G')$.
2. $\forall A \in V',\ \exists w \in T^*,\ A \Rightarrow_{G'}^* w$.

［**証明**］. 次の算譜は，明らかに終端記号を生む変数の集合 $\{A \in V \mid (A, w) \in P, w \in T^*\}$ から初めて，そこへバトンタッチする変数を徐々に増やしていく．こうして出来る閉包を V' と定めるような算譜である．

```
begin
    oldV = ∅;
    newV = {A ∈ V | (A, w) ∈ P, w ∈ T*}; プリミティブな変数のみまずは集める#
    while oldV ≠ newV do
        begin
            oldV = newV;
            newV = oldV ∪ {A ∈ V | (A, α) ∈ P, α ∈ (T ∪ oldV)*};
        end
    V' = newV;
end
```

とし，$P' = P \cap (V' \times (T \cup V')^*)$ とすれば良い．なお，$S \in V'$ は，$L(G) \neq \emptyset$ より，ある $w \in T^*$ が存在して $S \Rightarrow_G^* w$ であることから従う．

こうして定まる G' は 2 条件を満たす． □

補題 2.2.3 (開始記号から到達可能でない文字の除去算譜)**.** $G = (V, T, P, S)$ を文脈自由文法とする．次の 2 条件を満たす文脈自由文法 $G'' = (V'', T'', P'', S)$ を構成する算譜が存在する．

1. $L(G) = L(G'')$.
2. 任意の記号 $X \in V'' \cup T''$ について，開始記号 S から導出可能な文形式であって，V を含むものが存在する：$\forall X \in V'' \cup T'',\ \exists \alpha, \beta \in (V'' \cup T'')^*,\ S \Rightarrow_{G''}^* \alpha X \beta$.

［**証明**］. 次の算譜は，開始記号 S から初めて，閉包に到達するまで順次到達可能な文字を増やしていく算譜である．

```
    begin
        oldV = ∅;
        newV = {S};
        T'' = ∅;
        while oldV ≠ newV do
            begin
                oldV = newV;
                newV = oldV ∪ {B ∈ V | 記号 B を含むαと A ∈ oldV が存在して，(A, α) ∈ P};
                T'' = T'' ∪ {a ∈ T | 記号 a を含むαと A ∈ oldV が存在して，(A, α) ∈ P};
```

```
10                end
11            V′ = newV;
12        end
```

$P'' = V'' \times (V'' \cup T'')^*$ として定めた $G = (P'', V'', T'', S)$ は主張の条件を満たす．□

定理 2.2.4 (無駄な記号の除去算譜)**.** G を $L(G) \neq \emptyset$ を満たす文脈自由文法とする．次の 2 条件を満たす文脈自由文法 $\hat{G}$ を構成する算譜が存在する．

1. $L(G) = L(\hat{G})$.
2. $\hat{G}$ は無駄な記号を持たない．

［証明］. 補題 2.2.2 と補題 2.2.3 との算譜の合成を与えれば良い．□

2.2.2　空語生成規則

次は空語生成規則 $A \to \epsilon$ は，他の規則に埋め込めることを考える．

定義 2.2.5 (ϵ-production, nullable)**.** $A \to \epsilon$ の形の生成規則を，**ϵ-生成規則**という．文脈自由文法 G にこの生成規則が存在するとき，変数 A は**空語に潰せる**という．

定理 2.2.6. 文脈自由文法 $G = (V, T, P, S)$ に対して，次の 2 条件を満たす文脈自由文法 $G' = (V', T', P', S')$ を生成する算譜が存在する．

1. $L(G') = L(G) \setminus \{\epsilon\}$.
2. G' は無駄な変数も空語生成規則も持たない．

［証明］. 3 段階に分けて考える．

1. まず，次の算譜で空語に潰せる変数からなる集合 $\mathrm{NULL} \subset V$ を確定させる．

```
1             begin
2                 oldNULL = ∅;
3                 newNULL = {A ∈ V | (A, ϵ) ∈ P};
4                 while oldNULL ≠ newNULL do
5                     begin
6                         oldNULL = newNULL;
7                         newNULL = oldNULL ∪ {A ∈ V | ∃α ∈ oldNULL*, (A, α) ∈ P};
8                     end
9                 NULL = newNULL;
10            end
```

2. 次に，P' を，P の各生成規則 $A \to X_1 \cdots X_n$ に対して，新たに次を満たす規則 $A \to \alpha_1 \cdots \alpha_n$ を追加することで定める．
   (1) $X_i \notin \mathrm{NULL}$ ならば $\alpha_i = X_i$.
   (2) $X_i \in \mathrm{NULL}$ ならば $\alpha_i = X_i$ または $\alpha_i = \epsilon$.
   (3) $\exists i \in [n]$, $\alpha_i \neq \epsilon$ である．
   これらは各 $(A, X_1 \cdots X_n) \in P$ に対して有限個なので，P' も有限集合．また $G'' = (V, T, P', S)$ の時，次の 2 条件は同値であることが帰納法により確認できる．
   (1) $A \Rightarrow_{G''}^* w$.
   (2) $w \neq \epsilon$ かつ $A \Rightarrow_G^* w$.
   従って $L(G) = L(G'')$ である．
3. 定理 2.2.4 の算譜より G' を得ると，これは定理の条件を満たす．

□

2.2.3　単一生成規則の除去

定義 2.2.7 (unit production, nonunit production)**.** $A, B \in V$ を用いて $A \to B \in P$ と表される生成規則を**単一生成規則**という．そうでないものを**非単一生成規則**という．

定理 2.2.8. 文脈自由文法 $G = (V, T, P, S)$ が $\epsilon \notin L(G)$ であるとする．L は無駄な記号がなく，空語生成規則と単一生成規則を持たない文脈自由文法によっても生成される．

［証明］.

構成　定理 2.2.6 により得られる文脈自由文法を改めて $G = (V, T, P, S)$ とし，次を満たす生成規則の集合 P' を持った文脈自由文法 $G' = (V, T, P', S)$ を考える．

1. 全ての非単一生成規則を P' に入れる．
2. 任意の空語生成規則でない単一生成規則の合成は $A \Rightarrow_G^* B$ $(A, B \in V)$ と表せる．この時，B から始まる非単一生成規則 $B \to \alpha$ に対して，$A \to \alpha$ を P' に追加する．

検証　P' はうまく有限で閉じている．実際，$A \Rightarrow_G^* B$ をチェックする算譜が存在するのは，G が空語生成規則を持たないからである．これは必ず単一生成規則の合成となり，その総数は有限である．こうして確かに G' を計算可能に得る．G' は無駄な記号がなく，空語生成規則と単一生成規則を持たない文脈自由文法で確かに $L(G) = L(G')$ である．

□

2.3　Chomsky 標準形

二分木化 (binarization)

前節で開発した「無駄な記号がなく，空語生成規則と単一生成規則を持たない文脈自由文法」を，もう一つ理論的に扱いやすいものへと系をたてる．これは生成規則を理論的に扱いやすいように，有限のものを 2 に還元したものである．その分退避用の変数がたくさん生成される．実際，構文木は，葉を除けば二分木となっている．

定理 2.3.1 (Chomsky normal form)**.** 空語を含まない文脈自由言語 $\epsilon \notin L$ は，次の形をした生成規則しか持たない文脈自由文法 $G = (V, T, P, S)$ が存在して，これによって生成される．

1. $A \to BC$　$(A, B, C \in V)$.
2. $A \to a$　$(a \in T)$.

［証明］. 3 段階に分けて考える．

1. 定理 2.2.8 より，L を生成する無駄な記号がなく，空語生成規則と単一生成規則を持たない文脈自由文法 $G_1 = (V, T, P, S)$ が存在する．この P の生成規則は，次のいずれかである．
   終端記号を単一生成する規則　$A \to a$ $(A \in V, a \in T)$.
   記号を非単一生成する規則　$A \to X_1 \cdots X_m$ $(m \geqslant 2)$.
2. 終端記号生成は必ず単一になされるように整形する．即ち，非単一生成の中に終端記号が現れないようにする．文脈自由文法 $G_2 = (V', T, P', S)$ を次のように定めると，$L(G_1) = L(G_2)$ となる．各非単一生成規則 $\to X_1 \cdots X_m \in P$ に対して，次を満たす生成規則と変数を追加したものを P', V' とする．
   (1) $X_i = a$ $(a \in T)$ の時，新たな変数 C_a を追加して生成規則 $C_a \to a$ を P' に追加する．
   (2) 生成規則中の X_i を全て終端記号を単一生成する変数 C_a に置き換えて得られる生成規則を $A \to B_1 \cdots B_m$ とし，P' に追加する．
3. 非単一生成は終端記号を生まないようになったから，これを二元化する．$m \geqslant 3$ を満たす P' の生成規則 $A \to B_1 \cdots B_m$ について，新たな変数 $D_1, \cdots, D_{m-2}$ を導入し，

$$A \to B_1 D_1, \qquad D_1 \to B_2 D_2,$$

$$\vdots \qquad\qquad \vdots$$
$$D_{m-3} \to B_{m-2}D_{m-2}, \qquad D_{m-2} \to B_{m-1}B_m.$$

に分解して置き換える．こうして得る生成規則の集合を P''，変数の集合を V'' とすれば，$G_3 := (V'', T, P'', S)$ が定理の条件を満たす．

□

2.4　文脈自由言語の性質

Ogden の補題の $n' = N$ とした場合の退化した姿が反復補題であるが，こちらは直感がつかみやすい．文法規則の数に対して，十分に長い文字列を考えると，そこには文法規則の有限生に起因する反復許容性があるはずである．これを用いて文脈自由言語の必要条件を述べたのが Ogden の補題である．
Ogden の補題により，うまく算譜を見つけてこれば，文脈自由言語でないことを見破ることができる．その使用法は計算と場合分けで済む極めて煩瑣にテクニカルなもので，ここまで落とし込めるまでの祈りが莫大である．

2.4.1　反復補題 / Pumping lemma / uvwxy 定理

L が文脈自由言語であるという述語を CFL(L)，記号列 x が「反復可能」であることを Pump(x) と表すと，次の主張を反復補題という：

$$\mathrm{CFL}(L) \to \exists N \geqslant 1,\ \forall x \in L,\ [|x| \geqslant N \to \mathrm{Pump}(x)].$$

これは対偶が強く応用可能である：

$$\forall N \geqslant 1,\ \exists x \in L,\ [|x| \geqslant N \wedge \neg\mathrm{Pump}(x)] \to \neg\mathrm{CFL}(L).$$

この証明は Chomsky 標準形による二分木への翻訳があるために，極めて見通しが良い．これがプログラミングである．

定理 2.4.1 (pumping lemma)**.** 文脈自由言語 L に対して，定数 $N \geqslant 1$ が存在し，任意の $|z| > N$ を満たす記号列 $z \in L$ に対して，次の 3 条件を満たす分解 $z = uvwxy$ $(u, v, w, x, y \in L \cup \{\epsilon\})$ が存在する．

1. $|vx| \geqslant 1$.
2. $|vwx| \leqslant N$.
3. $\forall n \in \mathbb{N},\ uv^n wx^n y \in L$.

［証明］. 定理 2.3.1 より，$L \setminus \{\epsilon\}$ を生成する Chomsky 標準形の文脈自由文法が存在するから，これを $G = (V, T, P, S)$ とする．次が成り立つ．

(C) 長さ $2^{i-1}+1$ 以上の語 $w \in L$ の G-導出木には，根から葉に至る長さ（edge 数）が $i+1$ 以上の道が存在する．

対偶「道の長さの最大値が i 以下ならば，単語の最大長は 2^{i-1} である.」を考えれば明らか[†1]．そこで $k := |V|$ とし，$N := 2^k$ と定めると，任意の $|z| \geqslant N = 2^k$ を満たす語 z の導出木 T には，長さ $k+1$ のものが存在する．この道の通るノードの列を $v_1, \cdots, v_m, v_{m+1}$ $(m \geqslant k+1)$[†2]とし，そのラベルの列を $A_1, \cdots, A_m, a$ $(A_1 = S, a \in T)$ とする．すると，$m \geqslant k+1 = |V|+1$ より，鳩の巣原理から $A_{m-k}, A_{m-k+1}, \cdots, A_m$ の間に重複して現れている記号が存在する．その位置を i, j $(m-k \geqslant i < j \geqslant m)$ とする．v_i を根とする部分木を T_1，v_j を根とする部分木を T_2 とすると，$T_2 \subset T_1 \subset T$ で，T_2 で導出される終端記号列を $w \in L$ とすると，$v, x \in L \cup \{\epsilon\}$ が存在して vwx が T_1 が導出する記号列である．$m-k \geqslant i < m$ より，T_1 に存在する道の長さは最大で $k+1$ だから，$|vwx| \leqslant 2^k = N$．また，$j > i$ より，$|v| \geqslant 1 \vee |x| \geqslant 1$ である．$|vwx| \leqslant N, |z| \geqslant N$ より，$u, y \in L \cup \{\epsilon\}$ が存在して，$z = uvwxy$ と表せる．

[†1] 整数値しか取らないので，$|z| > 2^{i-1} \Leftrightarrow |z| \geqslant 2^{i-1}+1$.
[†2] 道の長さとノードの数にズレがある．植木算.

この時，状況を整理すると，

$$S \Rightarrow^* uA_iy, \qquad A_i \Rightarrow^* vA_jx, \qquad A_j \Rightarrow^* w$$

であるが，$A := A_i = A_j$ であるから，任意の $n \geqslant 0$ に対して，$S \Rightarrow^* uv^nwx^ny$ となり，導出可能である．従って，$uv^nwx^ny \in L$. □

所感 2.4.2 (Turing pattern?)**.** 反復補題とは，有限的な生成規則に起因する極めて大域的な模様か．Turing pattern とは違うのか．あまりに長いと，必ず同じ変数が再出現するので，そこに反復可能性が生ずる，という自然の摂理である．

系 2.4.3. $L \subset \Sigma^*$ を無限な自由文脈言語とする．ある正整数 $K > 0$ が存在して次を満たす：

$$\forall n \geqslant 1,\ \exists s_n \in L,\ [\, nK \geqslant |s_n| \geqslant (n+1)K \,].$$

［証明］. 定理**??**と同様の算譜で，$L = L(G)$ を満たす Chomsky 標準形の文脈自由文法 G をとり，$N = 2^{|V|}$ と定める．L は無限集合であるから，ある $z \in L$ が存在して $|z| \geqslant N$ を満たす．すると，反復補題（定理 2.4.1）を満たす分解 $z = uvwxy$ が存在する．この時，$|vx| \geqslant 1$ であるので，ある $c \geqslant 1$ が存在して，$|uwy| \leqslant c|vx|$ を満たす．この数を $K := c|vx|$ とすれば，各 $n \geqslant 1$ に対して $s_n := uv^{nc}wx^{nc}y$ とすることで，

$$\begin{aligned} nK = nc|vx| \leqslant |s_n| &= |uwy| + nc|vx| \\ &\leqslant c|vx| + nc|vx| = (n+1)c|vx| = (n+1)K. \end{aligned}$$

□

系 2.4.4. 次の言語は文脈自由言語ではない．

1. $\{a^nb^nc^n \mid n \geqslant 1\}$.
2. $\{a^mb^mc^n \mid 1 \geqslant m \geqslant n\}$.
3. $\{a^kb^lc^n \mid 1 \geqslant k \geqslant l \geqslant m\}$.
4. $\{a^mb^nc^md^n \mid m, n \geqslant 1\}$.
5. $\{ww \mid w \in \{a,b\}^*\}$.
6. $\{a^nb^nc^m \mid n \geqslant m \geqslant 2n\}$.
7. $\{a^mb^n \mid n = m^2\}$.

［証明］. 背理法による．

1. $L = \{a^nb^nc^n \mid n \geqslant 1\}$ を文脈自由言語とする．反復補題 2.4.1 より，十分大きな $N > 1$ について，$z = a^Nb^Nc^N \in L$ には，分解 $z = uvwxy$ であって，$|vx| \geqslant 1$，$|vwx| \leqslant N$，$\forall n \in \mathbb{N}$, $uv^nwx^ny \in L$ を満たすものが存在する．$|vwx| \leqslant N$ に注意すると，次の 2 つの場合に場合分けして考えられる．
   $vx \in \{a\}^* \vee vx \in \{b\}^* \vee vx \in \{c\}^*$ のとき　$n = 0$ とすると，それぞれ a, b, c の個数が他の個数より 1 個以上減るので，$uwy \notin L$.
   $vx \in \{a,b\}^* \vee vx \in \{b,c\}^*$ のとき　$n = 0$ とすると，同様に a, b, c の個数は不一致を起こす．
   どの場合も矛盾．よって，$L = \{a^nb^nc^n \mid n \geqslant 1\}$ は文脈自由言語ではない．
5. $L = \{ww \mid w \in \{a,b\}^*\}$ を文脈自由言語と仮定する．反復補題 2.4.1 より，十分大きな $N > 1$ が存在して，a が N 個続いた後に b が N 個続く文字列を $w_0 = a \cdots ab \cdots b$ としたときに $z = w_0w_0 \in L$ と表される記号列 $|z| \geqslant N$ も，$z = uvwxy$ に分解し，$|vx| \geqslant 1$，$|vwx| \leqslant N$，$\forall n \in \mathbb{N}$, $uv^nwx^ny \in L$ を満たす．$|vxy| \geqslant N$ より，次の 2 つの場合に分けられる．
   $vwx \in \{a\}^* \vee vwx \in \{b\}^*$ のとき　$|y| \geqslant 2N$（vwx が z の中央よりも左側にある）としても一般性は失われない．$n = 0$ としたとき，a が N 個続くというパターン，または b が N 個続くというパターンが左半分で二度と現れないので，$uwy \notin L$.
   $vwx \in \{a,b\}^*$ のとき　$|y| \geqslant N$ としても一般性は失われない．$n = 0$ としたとき，a が N 個続くというパターンは右半分では現れない，または b が N 個続くというパターンが左半分で現れないので，$uwy \notin L$ である．
   どの場合も矛盾．よって，L は文脈自由言語ではない．
7. $L = \{a^mb^{m^2} \mid m \in \mathbb{N}\}$ を文脈自由言語と仮定する．L は無限集合より，系 2.4.3 により，ある正整数 $K > 0$ が存在して，$\forall n \geqslant 1$, $\exists s_n \in L$, $nK \leqslant |s_n| \leqslant (n+1)K$ が成り立つ．いま，$|a^mb^{m^2}| = m(m+1)$ より，$m > K$ ととれば，

$K^2 < |a^m b^{m^2}|$ が成り立つ．このとき，$nK \leqslant |a^m b^{m^2}| \leqslant (n+1)K$ を満たす n について，$(n+1)K \leqslant |s_{n+1}| \leqslant (n+2)K$ を満たす $s_{n+1} \in L$ は存在しない．なぜならば，$a^m b^{m^2}$ の次に長い記号列は $a^{m+1}b^{(m+1)^2}$ であり，その長さの差は $|a^{m+1}b^{(m+1)^2}| - |a^m b^{m^2}| = (m+1)(m+2) - m(m+1) = 2(m+1) > 2K$ であるため，$(n+2)K < |a^{m+1}b^{(m+1)^2}|$ が成り立つからである．よって，L は文脈自由言語ではない．

□

所感 2.4.5. 証明は $N > 1$ を取った後に，任意に取って良い $|z| > N$ を馬鹿でかく取ること．すると場合分けが簡略化され，示しやすくなる．技工は要らないというのがこの反復補題**??**の本質なのだから，馬鹿でかく取って技工を使わないのが綺麗．

命題 2.4.6. 言語 $\{x \in \{0,1\}^* \mid [x]_{10}$は 3 の倍数 $\}$ は文脈自由言語である．

［証明］．$V = \{S, S^-, S^+\}, T = \{0,1\}$ とし，

$$P = \left\{ \begin{array}{lll} S \to 0, & S^- \to 1, & S^+ \to S^-0, \\ S \to S0, & S^- \to S^+0, & S^+ \to S^+1, \\ S \to S^-1, & S^- \to S1 & \end{array} \right\}$$

とすると，文脈自由文法 $G = (V, T, P, S)$ は $L(G) = \{x \in \{0,1\}^* \mid [x]$ は 3 の倍数 $\}$ となる．ただし，$[01] = [1] = [00\cdots01] = 1$ とした．□

所感 2.4.7. 例 1.2.5 の通り，正則言語でもある．設計していて思ったが，変数 V が状態っぽい．

2.4.2　Ogden の補題

反復補題は Ogden の補題の，必ず特定位置を最大限とった $n' = n$ の特別な場合である．確かに反復補題の主張は少し人工的で，Chomsky 標準形に整えた場合の二分木の消息に寄り添い切って居なかった．**鳩の巣原理**による大域的模様という主張の核心は変わらず，算譜を少し変更するだけで済む．

定理 2.4.8 (Ogden)**.** 文脈自由言語 L に対して，定数 $N > 1$ が存在し，任意の $|z| \geqslant N$ を満たす記号列 $z = z_1 \cdots z_n$ $(n \geqslant N)$ と，任意の特定位置 $d_1, \cdots, d_{n'}$ $(N \leqslant n' \leqslant n, 1 \leqslant d_1 < \cdots < d_{n'} \leqslant n)$ に対して，次の 3 条件を満たす分解 $z = uvwxy$ $(u, v, w, x, y \in L \cup \{\epsilon\})$ が存在する．

1. vx は少なくとも 1 つの特定位置を含む．
2. vwx は高々 N 個の特定位置を含む．
3. $\forall n \in \mathbb{N}$, $uv^n wx^n y \in L$.

［証明］．$L \setminus \{\epsilon\}$ を生成する Chomsky 標準形の文脈自由文法を $G = (V, T, P, S)$ とする．反復補題 2.4.1 の場合と同様に，$k := |V|$ とし，$N := 2^k$ と定める．ここで，$|z| = n \geqslant N = 2^k$ を満たす語 z の導出木 T を考える．特定位置 $d_1, \cdots, d_{n'}$ $(N \leqslant n' \leqslant n)$ に対応する葉のうち，根からの距離が最大であるものを 1 つ取り，その根からの道のノードの列を $v_1, \cdots, v_m, v_{m+1}$ とする．このとき，この道の長さが $k+1$ 以上，従って $m \geqslant k+1$ であることを示せば，以降は反復補題 2.4.1 と全く同じ設定へと還元できた事になる．

反復補題 2.4.1 の証明における条件 (C) の対偶より，長さ k 以内の部分木で到達可能な終端記号は，2^{k+1} 文字よりも少なく，特に N より少ない．従って $N \leqslant n'$ より，特定位置 $d_1, \cdots, d_{n'}$ に対応する葉のうち，根からの距離が $k+1$ 以上であるものは必ず存在する．□

系 2.4.9. 次の言語は文脈自由言語ではない．

1. $\{0^i 1^j 0^k \mid i, j, k \in \mathbb{N}, j = \max(i, k)\}$.
2. $\{a^n b^n c^i \mid i, n \in \mathbb{N}, n \neq i\}$.

［証明］．背理法で示す．

1. 十分大きな $N > 0$ について，文字列 $0^N 1^N 0^N$（長さ $n = 3N$）を考える．左から読んで最初のブロック 0^N の全てを特定位

置とする（$n' = N$）．これについて，Ogden の補題 2.4.8 より，分解 $z = uvwxy$ が存在する．$|vwx| \leqslant N$ より，$|vwx| \subset 0^N$ である．$|vx| \geqslant 1$ より，n を十分大きくすると，$uv^n wx^n y = 0^i 1^j 0^k$ における i の数が $j = \max(i,k)$ であるはずの j の数より大きくなり，矛盾．

2. 記号列 $a^n b^n c^{n!}$ を考える．

□

所感 2.4.10. どうしてこんな議論をしているのかわからないほどに，ただ単純な計算と場合分けに議論が落とし込まれているのが Ogden の補題である．

2.5　所属性判定問題

所属判定問題という形で，形式言語論が計算可能性理論に接続する．また，コンパイラの原理の一つでもある．議論としては，自然数上の原始再帰（部分）関数による計算可能性の定義に繋がる．文脈自由言語は帰納的可算言語なので，計算可能である．問題は計算量になる．
導出は構文木の構造があるが，構文木の探索は三角チャートの構造を持つ．なんとなく，データ構造自体が計算と同義なのではないかと思うようになった．圏論の精神．

定義 2.5.1 (membership problem)**.** 文脈自由文法 G に対して，二項関係 $w \in L(G)$ を判定させる問題を**所属判定問題**という．

議論 2.5.2 (CKY algorithm)**.** 全ての構文木を列挙すると，構文木の数は指数爆発する（カタラン数と呼ばれる組み合わせ数学の対象）．CKY アルゴリズムと呼ばれるチャートを下から埋める動的計画法により余計な計算が減る．このときに使うデータ構造が chart / 三角テーブルである[†3]．なんとなく Jordan 標準形の議論を思い出す添字の付け方であった．各セル (x,y) は区間 $(x,y]$ の部分記号列を生成する構文木（今回は根のラベルのみ）を保存し，$(0,n)$ が構成できたならば構文木の存在を構成できたことになる．

1. まず，チャートの一段目 $\mathtt{cell}(i,i+1)$ には終端文字 $(i,i+1] = x_{i+1}$ を生成する変数からなる集合を格納する．
2. (x,y) を作るには，垂直な直線 $(x,k),(k,y)$ から作る．これは下から埋めていくこととなる．各 k について，$\mathtt{cell}(x,k)$ の変数 X と $\mathtt{cell}(k,y)$ の変数 Y であって，これらを導出する $Z \to XY$ がないか探し，あれば新たに $\mathtt{cell}(x,y)$ に格納していく．
3. こうして，全体空間を格納する cell が空でなければ良い．

このように葉から解析を始めることをボトムアップという．CYK は Cocke-Younger-Kasami の略（それぞれ、RISC の先駆と言われる 801 などでも知られるジョン・コック、Daniel Younger、嵩忠雄である）。

注 2.5.3 (chart parser)**.** chart のデータ構造により組み合わせ爆発を防ぐ方法一般を chart parser という．

> チャートパーサ（英: Chart parser）は、自然言語などの曖昧な文法に向いた構文解析器の一種である。動的計画法を用い、中間的かつ仮説的な結果をチャート（chart）と呼ばれるデータ構造に格納しておき、再利用する。これによりバックトラッキングを省き、同時に組合せ爆発を防ぐ。チャートパーサは Martin Kay が開発した。[†4]

- ビタビアルゴリズムを変形したものを用いることが多い。
- アーリー法はチャートパーサの一種であり、計算言語学での構文解析に主に使われる。
- CYK 法もチャートパーサの一種である。
- 左隅型解析法
- 一般化 LR 法：1986 年、冨田勝が発表した。冨田法、並列構文解析法とも呼ばれる。

注 2.5.4. より限定的な文脈自由文法のクラスに限れば，より高速な認識アルゴリズムが存在する．

[†3] 第一変数と第二変数を直交基底として取ればいいのにな．すると直交するのに．なんで三角にするのだろう．
[†4] wikipedia

- LR 法：自然言語の非決定的で曖昧な性質に対処できない
- LL 法

コンパイラ学ではこちらが議論の中心になるが，自然言語には使えない．

定理 2.5.5 (CKY algorithm)**.** 文脈自由文法 G と記号列 $w \in T^*$ に対して，$w \in L(G)$ は計算可能／決定可能である（判定する算譜が存在する）．

［証明］.

$w = \epsilon$ のとき　無駄な変数も空語生成規則も持たない文脈自由文法の存在定理 2.2.6 の算譜より，$S \in \mathtt{NULL}$ か計算できる．

$w \neq \epsilon$ のとき　入力を $w = x_1, \cdots, x_n$ $(x_1, \cdots, x_n \in T)$ とおく．Chomsky 標準形の文脈自由文法の存在定理 2.3.1 より，$L(G') = L(G) \setminus \{\epsilon\}$ となる Chomsky 標準形の文脈自由文法 $G' = (V, T, P, S)$ を計算できる．次は $w \in L(G')$ を判定する算譜である．

```
1   begin
2       for i = 0 to n − 1 do ＃チャートの 1 段目の初期化
3           cell(i,i+1) = {A ∈ V | A → x_i ∈ P};
4       for i = 2 to n do
5           for x = 0 to n − i do
6               y = x + i;
7               for k = x+1 to y − 1 do ＃分割位置
8                   for X ∈ cell(x,k)
9                       for Y ∈ cell(k,y)
10                          for Z ∈ V
11                              if Z → XY ∈ P then cell(x,y) = cell(x,y) ∪ Z;
12      if S ∈ cell(0, n) then w ∈ L(G') else w ∉ L(G')
13  end
```

□

所感 2.5.6. 最悪計算量は $O(n^3)$ $(n = |w|)$．まあまあ遅いが，全ての文脈自由文法に対応できるものでは最速に近い．i, x, k のループ数（場合の数）は $O(|w|^3)$（−1 などの影響は省けるという意味で）．X, Y, Z のループ数（場合の数）は $O(|V|^3)$．よって，時間計算量は $O(|w|^3|V|^3)$.

Chomsky 標準形

Chomsky 標準形では，終端記号以外の単項生成を許さない．これにより，一つの cell 内での自己生成を考えなくて良いので，アルゴリズムが単純になる．

2.6　有限オートマトンと正則文法

正則言語が，文脈自由言語の真部分集合であることを見る．そのために用いるのは，線型文法による特徴付けである．定理 2.6.2 にて，

1. DFA の右線型文法への還元算譜.
2. 右線型文法の，二分木化を経た ϵ-NFA への還元算譜.
3. DFA の，NFA への反転算譜 $^R : [\mathrm{DFA}] \to [\mathrm{NFA}]$.

の 3 つを導入して，世界を繋ぐ．こうして，オートマトンも形式科学の研究対象として随分と取り込んだことになる．

定義 2.6.1 (right-linear grammar, left-linear grammar, regular grammar)**.** $A, B \in V$, $w \in T^*$ とする．

1. 全ての生成規則が $A \to w \mid wB$ の形である文脈自由文法 G を，**右線型文法**という．

2. 全ての生成規則が $A \to w \mid Bw$ の形である文脈自由文法 G を，**左線型文法**という．
3. 右線型文法と左線型文法のことを，併せて**正則文法**という．

定理 2.6.2 (正則言語の特徴付け)**.** 言語 $L \subset \Sigma^*$ に対して，次の 3 条件は同値．

1. L は正則．
2. L は右線型文法によって生成される．
3. L は左線型文法によって生成される．

［証明］**.**

(1)⇒(2)　L を正則とする．すると，これを受理する DFA $M = (Q, \Sigma, \delta, q_0, F)$ が存在して $L = L(M)$ となる．これに対して右線型文法 $G = (V, T, P, S)$ を，$V = Q, T = \Sigma, S = q_0$ とし，P を

1. 各 $\delta(q, a) = p$ に対して，$q \to ap \in P$ とし，
2. 各 $q \in F$ に対して，$q \to \epsilon \in P$ とする．

すると，$L(G) = L(M) = L$ となる．

(2)⇒(1)　L は右線型文法によって生成されるとする．すると，右線型文法 $G = (V, T, P, S)$ が存在して $L = L(G)$ である．

1. まず，右線型文法を二分木化する．G の生成規則は，$a_1, \cdots, a_m \in T$ を用いて $A \to a_1 \cdots a_m B \mid a_1 \cdots a_m$ $(m \in \mathbb{N})$ と表せる．この $m \geqslant 2$ の場合について，前者は新たな変数 $C_1, \cdots, C_{m-1}$ を導入して，m 個の生成規則

$$A \to a_1 C_1, \qquad C_1 \to a_2 C_2, \qquad \cdots \qquad C_{m-1} \to a_m B,$$

で置き換え，後者は新たな変数 $D_1, \cdots, D_{m-1}$ を導入して，m 個の生成規則

$$A \to a_1 D_1, \qquad D_1 \to a_2 D_2, \qquad \cdots \qquad D_{m-1} \to a_m,$$

で置き換えて得られる右線型文法を $G' := (V', T, P', S)$ とする．このとき，P の元は $A \to a_1 \cdots a_m B \mid a_1 \cdots a_m$ $(m = 0, 1)$ と表せる状態だから，書き下せば次の 4 通りのいずれかの形をしている．

(a)　$A \to aB$ $(a \in T)$,
(b)　$A \to B$,
(c)　$A \to a$,
(d)　$A \to \epsilon$.

2. この二分木化された右線型文法 G' を ϵ-NFA $M = (Q, \Sigma, \delta, q_0, F)$ に変換する．まず新たな終了状態 f を追加して $Q = V' \cup \{f\}$, $F = \{A \in V' \mid A \to \epsilon\} \cup \{f\}$ とする．$\Sigma = T, q_0 = S$ で，遷移関係 $\delta \subset (Q \times \Sigma) \times Q$ は次のように定める．

(a)　$A \to aB$ のとき，$(A, a, B) \in \delta$.
(b)　$A \to B$ のとき，$(A, \epsilon, B) \in \delta$.
(c)　$A \to a$ のとき，$(A, a, f) \in \delta$.

すると，$L(M) = L(G') = L(G) = L$ を得る．

(1)⇔(3)　同様の議論を L の代わりに L^R について行ったものが，L についての主張 (1)⇔(3) の証明を与える．ただし，オートマトンの反転，即ち，オートマトン $M = (Q, \Sigma, \delta, q_0, F)$ に対して，$L(M)^R$ を受理するオートマトンは，非決定性として，次のように構成する．開始状態 q_0 を唯一の終了状態とし，新たな開始状態 q_0' を追加する．$M^R = (Q \cup \{q_0'\}, \Sigma, \delta', q_0', \{q_0\})$ を

(a)　$\delta(p, a) = q$ のとき，$(q, a, p) \in \delta'$ とし，
(b)　$q \in F$ のとき，$(q_0', \epsilon, q) \in \delta'$ として，実質的な開始状態とする．

□

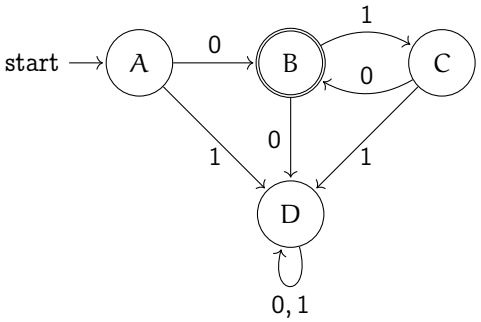
所感 2.6.3 (3 つの同型対応)**.**

1. (1)⇒(2) で極めて簡明なオートマトンの右線型文法への翻訳算譜を作った．M に ϵ-遷移がない場合は，右線型文法のさらに真のクラスになるような文法に翻訳されている（終端文字を単一生成する規則は ϵ-生成規則のみ)．
2. (2)⇒(1) はまずオートマトンのために二分木化するところまでは良い．右線型文法 G の生成規則は，$a_1, \cdots, a_m \in T$ を用いて $A \to a_1 \cdots a_m B \mid a_1 \cdots a_m$ $(m \in \mathbb{N})$ と表せるが，$m \geqslant 2$ の場合を全て $m = 0, 1$ の場合に還元すれば良い．ところが，

次は ϵ-NFA に変換する．たくさんのマシンを定義したので，ここに来て論理の自由さがありがたい．ϵ 生成規則は終了状態に至るが，そうでない終端文字の単項生成規則の場合は新たに作った終了状態 $f \in F$ に集めるというトリックが必要である．

3. (1)⇔(3) では，文法における右線型・左線型の反転に対する NFA への反転操作 M^R を定義した：$^R : [\mathrm{DFA}] \to [\mathrm{NFA}]$.

例 2.6.4 (DFA を右線型文法へ翻訳する)**.** 次のオートマトンは正則言語 $0(10)^*$ を受理する．



これと同じ言語を受理する右線型文法は，定理 2.6.2 の (1)⇒(2) の証明を参考にして，

$$A \to 0B \mid 1D \mid 0, \qquad B \to 0D \mid 1C,$$
$$C \to 0B \mid 1D \mid 0, \qquad D \to 0D \mid 1D.$$

となるが，最終処理場としての状態 D は**無駄な変数**だからこれを取り除く算譜（定理 2.2.4）が存在して，

$$A \to 0B \mid 0, \qquad B \to 1C, \qquad C \to 0B \mid 0.$$

となる．

所感 2.6.5. これは D に関連する全ての記述を消去しただけである，相対位相 $P \cap (\{A, B, C\} \times (\{A, B, C\} \cup T)^*)$ をとったみたいな．

2.7　Pushdown automata

極めて綺麗な理論である．DFA/NFA に無限長の stack のデータ構造を備えた機械とみなせる．データ構造の追加法によって，stack machine も registar machine もあり得る．stack の一番上の内容に依って，遷移の仕方を制御できる．また，純粋な stack 操作も遷移関係に定義しておくことができる，ある種自分で自分をプログラムできる．
PDA M の現状の表示を 3-組として定義し，そこに遷移関係を写し，$\vdash_M \subset Q \times \Sigma^* \times \Gamma^*$ を得る．この反射的推移閉包 $\vdash_M^*$ を計算することが，受理するかどうかの判定に等しい．

2.7.1　定義と動き

定義 2.7.1 (PDA: pushdown automaton)**.** 次の 6 つ組 $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$ をいう．

1. Q は状態からなる有限集合．
2. Σ は入力アルファベットの有限集合．
3. Γ は stack アルファベットの有限集合．
4. $\delta \subset (Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma) \times (Q \times \Gamma^*)$ は遷移関係．Γ は stack の top で，Γ^* とは，消去と，新たに書き込む文字は列であることを許容する．
5. $q_0 \in Q$ は初期状態．
6. $Z_0 \in \Gamma$ は初期記号．

注 2.7.2 (その他の抽象機械の設計法)**.** 有限オートマトンにふたつのスタックを接続することもでき、これは事実上チューリングマシンと等価な非常に強力なデバイスとなる。線形拘束オートマトンはプッシュダウン・オートマトンよりも強力だが、チューリングマシンよりは非力である。

定義 2.7.3 (ID: instantaneous description)**.** PDA の**時点表示**とは，次を満たす 3-組 (q, w, γ) をいう．

1. $q \in Q$ はヘッドの指し示す状態.
2. $w \in \Sigma^*$ は残りの入力文字列.
3. $\gamma \in \Gamma^*$ は stack 上の文字列.

定義 2.7.4 (遷移関係). PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$ の 2 つの時点表示の間の**遷移関係**と呼ばれる二項関係 $\vdash_M$ を，次のように δ から定義する：

$$(q, aw, X\gamma) \vdash_M (q', w, \Delta\gamma) :\Leftrightarrow ((q, a, X), (q', \Delta)) \in \delta, \qquad X \in \Gamma, \Delta, \gamma \in \Gamma^*$$
$$(q, w, X\gamma) \vdash_M (q', w, \Delta\gamma) :\Leftrightarrow ((q, \epsilon, X), (q', \Delta)) \in \delta.$$

遷移関係の反射的推移閉包を $\vdash_M^*$ と表す.

所感 2.7.5. PDA は必ず空の入力が δ に定義されていることに注意.

定義 2.7.6 (受理条件). PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$ が $w \in \Sigma^*$ を受理するとは，$(q_0, w, Z_0) \vdash_M^* (q, \epsilon, \epsilon)$ が成り立つことをいう.

所感 2.7.7. 入力を全て受け取った時に，stack が空になったら（なる受理列が存在すれば）受理とする．終了状態 $F \subset Q$ を定義する流儀もある．stack は必ず消費されなきゃいけないが，stack を空に消去する命令を追加することもできる.

2.7.2　CFG との等価生

定義 2.7.8 (Greibach normal form). 文脈自由言語 $G = (V, T, P, S)$ の全ての生成規則が次の 2 つのいずれかの形をしているとき，G を**グライバッハ標準形**という.

1. $A \to aB_1 \cdots B_m \quad (A \in V, a \in T, B_1, \cdots, B_m \in V \setminus \{S\})$.
2. $S \to \epsilon$.

左再帰が許されないという点で，右線型文法の $m \geqslant 2$ を一部許容した形である.

定理 2.7.9 (Greibach). 全ての文脈自由文法 G は等価なグライバッハ標準形の文法 G' に書換えることができる：$L(G) = L(G')$.

定理 2.7.10. 言語 L について，次の 2 条件が同値.

1. ある PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$ が存在してこれが認識する：$L = L(M)$.
2. ある CFG $G = (V, T, P, S)$ が存在して $L = L(G)$ である.

［証明］. PDA と CFG の間に対応関係を定めれば良い.

(1)⇒(2)　M の遷移関係 δ に対して，$((q, a, A), (q_1, B_1 \cdots B_m)) \in \delta$ のとき，任意の状態 $q_1, \cdots, q_{m+1} \in Q$ に対して，

$$(q, A, q_{m+1}) \to a(q, B_1, q_1)(q_1, B_2, q_2) \cdots (q_m, B_m, q_{m+1})$$

を P に追加し，各変数 (q, A, q_{m+1}) を V に追加して，P', V' を構成する．新たに追加された生成規則は，他の項 $(q_1, B_2, q_2) \cdots (q_m, B_m, q_{m+1})$ も消費しなければ終端文字列を生成できないが，これは stack を空にしないと受理されないことに対応する.

(2)⇒(1)　Greibach 標準形 2.7.9 の文法に対して，$A \to aB_1 \cdots B_m$ を $((q_0, a, A), (q_0, B_1 \cdots B_m)) \in \delta$ とする.

□

所感 2.7.11 (Greibach 標準形が全て). (1)⇒(2) では，状態という概念を全て変数概念へ退避しなければならない．新たに作った変数 (q, A, q') は，状態 q から q' へ遷移し，stack に A と書き込むことで受理し得る文字列を生成し得る変数として code されるのである．(2)⇒(1) では，左から終端記号に変換していき，残りの文字列を stack に入れておくのである．これは，Greibach 標準形 2.7.9 がすごいのである.

正則文法と文脈自由文法の違いは，stack というデータ構造が使える分だけ違う，というのである．有限オートマトンとは違って，今まで読んだ文字の情報を覚えておくことができるのである．ここに文字のパターン・繰り返しだけではなく，個数という概念が生まれる．

2.8　Chomsky 階層

$\alpha, \beta, \gamma \in (V \cup T)^*$，$A, B \in V$ は非終端記号，$a \in T$ は終端記号とする．

階層名	文法	言語	オートマトン	生成規則
タイプ 0	形式文法	帰納的可算言語 帰納言語	Turing 機械	$\alpha \to \gamma$
タイプ 1	文脈依存文法	文脈依存言語	線形拘束オートマトン	$\alpha A \beta \to \alpha \gamma \beta$
タイプ 2	文脈自由文法	文脈自由言語	プッシュダウンオートマトン	$A \to \gamma$
タイプ 3	正則文法	正則言語	有限オートマトン	$A \to a, A \to aB$

注 2.8.1 (文法の分類)**.** チョムスキー階層は、生成規則による終端および非終端記号からなる文字列の書き換えで定義される、生成文法と呼ばれる形式文法のクラスを軸に定義されている。具体的には、

0. ［**帰納的可算言語・部分決定性言語・チューリング受理性言語**］文字列のいかなる書き換えも許される制限なし文法（＝形式文法）がタイプ-0 であり、これを決定性のある（つまりチューリングマシンが常に停止する）言語に限定したクラスが帰納言語で、決定性言語またはチューリング決定性言語とも呼ばれる。これらの計算複雑性はそれぞれ複雑性クラス R と RE に対応する。つまり、与えられた文字列が、その言語の文字列である事を確かめるアルゴリズムを書ける言語の集合が帰納的可算言語で、その言語の文字列で無い事も確かめるアルゴリズムが存在する言語の集合が帰納言語である。
1. ［**文脈依存言語**］それぞれの生成規則が非終端記号ひとつのみを書き換える文脈依存文法がタイプ-1（文脈自由言語により近い所には、自然言語の研究から弱文脈依存言語というクラスが設定されている。）
2. ［**文脈自由言語**］文脈依存文法のうち前後の文字列に依存せず書き換える文脈自由文法がタイプ-2
3. ［**正規言語**］文脈自由文法のうち書き換えが終端記号一つまたは終端および非終端記号一つずつである正規文法がタイプ-3. 有限オートマトンが定める．

下方の文法クラスがそれぞれ上方の文法クラスすべての真部分集合となっているため、対応する言語も包含階層が成立する。なお、それぞれに対応するオートマトンもよく知られている。

Chomsky 階層とは，「計算可能」な言語の分類である．数式や C 言語のプログラム自体は文脈自由文法の範疇に入るデータ構造である．日本語も同じ範疇だと考えられる．一方で C 言語の変数は正則言語である．

注 2.8.2. Gerald Gazdar と Geoffrey Pullum は、一部に文脈自由的でない構造があるものの、自然言語の大部分は文脈自由であると指摘している [?, ?]。文脈自由でない部分とは、例えば、スイスドイツ語の cross-serial dependencies[?, ?] や、バンバラ語の畳語である [?, ?]。

歴史 2.8.3 (Avram Noam Chomsky 28-)**.** 生成文法を 1950 年代に提唱した．『「生成文法理論」を提唱して，言語学の革命的大転回を引き起こし，これを通じて人間の精神構造を解明するという野心的なプログラムを可能にした．これにより認知科学の成立を鼓舞し，その基盤を与えた．』貢献で 1988 年京都賞受賞．現在もマサチューセッツ工科大学教授．

2.8.1　形式文法

定義 2.8.4 (formal grammar)**.** 次の 4 つ組 $G = (V, T, P, S)$ を**形式文法**という．

- V は非終端記号からなる有限集合．
- T は終端記号からなる有限集合で V と交わらない．

- $P \subset (V \cup T)^* \cdot V \cdot (V \cup T)^* \times (V \cup T)^*$ は生成規則からなる有限集合.
- $S \in V$ は開始記号.

定義 2.8.5 (Recursively enumerable language)**.** 形式文法 $G = (V, T, P, S)$ が定める言語

$$L(G) := \{w \in T^* \mid S \Rightarrow_G^* w\}$$

を，**帰納的可算言語**という.

注 2.8.6. そのほかの同値な定義は

1. 形式言語のアルファベットから生成可能な全ての単語の集合 T^* のうち，帰納的可算な部分集合である.

などがある．名前の由来は，帰納的可算集合と言語 Σ^* との共通部分であるからである.

2.8.2　閉包属性の違い

0. Kleene 閉包 *，連結 $\cdot$，逆転 ^{R} ？？，$\cup, \cap$.
1. Kleene 閉包 *？，連結 $\cdot$，逆転 ^{R} ？？，$\cup, \cap$，補集合 $\Sigma^* \setminus -$.
2. Kleene 閉包 *，連結 $\cdot$，逆転 ^{R}，$\cup$（命題 2.1.8）．準同型 $\varphi(L)$，正則言語 D との積 $L \cap D$.
3. Kleene 閉包 *，連結 $\cdot$，逆転 ^{R}，$\cup, \cap$，差集合 $\setminus$，補集合 $\Sigma^* \setminus -$．shuffle $\vee$，商集合 $/$（第 1.8 節）.

第 3 章

Turing 機械

意味論の畳み込みが面白い．第 3.6 節では，全ての主体の意味論を削ぎ落として $\mathbb{N}$ と $\{0,1\}$ の 2 つをアルファベットとして採用する．また，tape の数も意味論的な都合であり，一つにまとめる還元算譜が存在する．有限回の手続きの列を計算というように，還元算譜が知能であるのかもしれない．

次に，形式科学的対象も同じく引き戻すのである．Turing 機械を符号化 $\{0,1\}^* \supsetneq \mathrm{CODE} \to \mathrm{TM}$ を通じて，$(w_i)_{i\in\mathbb{N}} : \mathbb{N} \to \{0,1\}^*$ を用いて自然数 $\mathbb{N}$ に引き戻す．こうして，2 つの次元の違う対象を同じ台集合の上で衝突させると，対角線論法から大枠への言及を得る．

こうして，形式科学的対象（高々可算個）を用いて，問題（連続体濃度）に射を入れていく．大抵の射の入れ方は等価になって，これを Church のテーゼという．これが計算理論の始まりなのだろう．

3.1 定義

PDA と同様，tape に書いてある内容を元に遷移が定まっているか，未定義であるような機械である．stack を対象にしたデータ構造の上を自由に行き来できる点が違う．線形拘束オートマトン (Linear Bounded Automaton) の場合でさえ PDA よりも計算力が真に強い（が Turing 機械より真に弱い）．線形拘束オートマトンは文脈依存言語を受容する．
帰納的可算，帰納的，計算可能，決定可能が大集合する母なる大地である．Zorn の補題とは計算可能性についての命題であったのか？

定義 3.1.1 (Turing machine, tape symbol, blank symbol)**.** チューリング機械とは，次の条件を満たす 7-組 $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ のことである．

1. Q は finite control の状態全体からなる有限集合．
2. Γ は tape 記号からなる有限集合．tape の cell に書き込むことが出来る動作の全体を表す．
3. $B \in \Gamma$ は空白記号．
4. $\Sigma \subset \Gamma \setminus \{B\}$ は出力アルファベット．[†1]
5. $\delta \subset ((Q \times \Gamma) \times (Q \times \Gamma \times \{L, R\}))$ は遷移を定める部分関数．
6. $q_0 \in Q$ は初期状態．
7. $F \subset Q$ は受理状態．

注 3.1.2. head の動きには N（動かない）を追加する流儀もあるが，空な L, R 動作で置き換えられる．

定義 3.1.3 (ID: instantaneous description / configuration)**.**

1. tape の左端を 1 とし，空白記号以外の記号が書かれている cell の右端を $n \in \mathbb{N}$ とすると，**テープの内容**を $X_1 \cdots X_n \in \Gamma^*$ と表す．$n = 0$ のとき，テープに書かれている文字は全て空白記号であることをいう．
2. Turing 機械の状態が q であり，ヘッドが i 番目の cell にあり，テープの内容が $X_1 \cdots X_n$ であるとき，この時の**時点表示**ま

[†1] 第 3.6 節では，これを入力アルファベットと完全に一致させることがミソとなる．

たは**計算状況**の 3-組と見做していたものを

$$X_1 \cdots X_{i-1} q X_i \cdots X_n$$

と表す．

定義 3.1.4 (move, calculation, halt)．ID が $X_1 \cdots X_{i-1} q X_i \cdots X_n$ であるとする（ただし，$i-1=n$ である時，$X_i = B$ である事になる）．ID の間の**動作**と呼ばれる二項関係を，次のように定義する．

1. $\delta(q, X_i) = (p, Y, L)$ $(1 < i \leqslant n+1)$ である時[†2]，
$$X_1 \cdots X_{i-1} q X_i \cdots X_n \vdash_M X_1 \cdots X_{i-2} q X_{i-1} Y X_{i+1} \cdots X_n$$
とする．ただし，$i = n+1$ で $Y = B$ であった場合，これは ID から取り除いておく．
2. $\delta(q, X_i) = (p, Y, R)$ $(1 \leqslant i \leqslant n+1)$ である時，
$$X_1 \cdots X_{i-1} q X_i \cdots X_n \vdash_M X_1 \cdots X_{i-1} Y p X_{i+1} \cdots X_n$$
とする．ただし，$i = n+1$ であった場合は $X_{i+1} = X_{n+2} = B$ はもちろん取り除くが，$Y = B$ である場合はこれは取り除けない．
3. 動作の列 $I_1 \vdash_M I_2 \vdash_M \cdots \vdash_M I_m$ を**計算**と呼ぶ．
4. また，動作 $\vdash_M$ を有限回繰り返して得られる関係を $\vdash_M^*$ と表し，これも**計算**と呼ぶ：$I_1 \vdash_M^* I_m$．
5. ID I について，$I \vdash_M J$ となる ID J が存在しない時，I **で停止している**という．

定義 3.1.5 (initial ID, accepting ID, accepting language)．Turing 機械 $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ ついて，

1. 入力 $w \in \Sigma^*$ に対して，ID $q_0 w$ を**初期 ID** または**初期計算状況**という．
2. $q \in F$ について，ID $\alpha_1 q \alpha_2$ $(\alpha_1, \alpha_2 \in \Sigma^*)$ を**受理 ID** または**受理計算状況**という．
3. 受理言語とは，次の集合をいう：
$$L(M) := \{w \in \Sigma^* \mid \exists p \in F,\ \exists \alpha_1, \alpha_2 \in \Sigma^*,\ [q_0 w \vdash_M^* \alpha_1 p \alpha_2]\}.$$

命題 3.1.6 (Turing 機械における停止の概念)．Turing 機械 $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ ついて，δ' を次のように定め，新たに Turing 機械 M' を考える．

$$\delta'(q, X) := \begin{cases} \delta(q, X), & q \notin F \text{ のとき}, \\ \text{undefined}, & q \in F \text{ のとき}. \end{cases}$$

このとき，次が成り立つ．

1. $L(M) = L(M')$．
2. $w \in L(M')$ であることと，w が停止する計算によって受理されることとは同値．

所感 3.1.7. 即ち，Turing 機械において，受理することと停止することとを等価にするような Turing 機械の還元算譜が存在するから，事実上等価に扱って良い，ということだろうか？

定義 3.1.8 (recursively enumerable set)．Turing 機械によって受理される言語を，**帰納的可算言語**という．言語であって，帰納的可算集合であるようなものだからである．Σ^* に関する相対位相．

定義 3.1.9 (recursive / computable / decidable set)．Turing 機械 $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ ついて，

1. 全ての入力 $x \in \Sigma^*$ に対して，M は停止するとき，M は**停止する Turing 機械**であるという．
2. 停止する Turing 機械によって受理される言語を**帰納的集合**という．
3. 集合 L については，帰納的であるとき[†3]，**決定可能**であるともいう．帰納的でないことを**決定不可能**ともいう．

[†2] $i = 1$ である時，head が tape から落ちるので，移動は定義されない．

[†3] 特性関数が計算可能であるという定義も存在する

3.2　Turing 機械による関数の計算

再帰的関数 (Kleene, 1952) の定義「初期関数から，合成・再帰的定義・最小化作用素による定義の 3 種を有限回施して得られる部分関数の閉包を再帰的（部分）関数という.」の Turing 機械による特徴付けを得る．帰納的関数は決定可能で，部分帰納的関数は半決定可能である．yes 以外の場合は停止するかもわからないクラスが，部分関数 $\mathbb{N} \to \mathbb{N}$ に対応する．

定義 3.2.1 (decidable / computable)**.** 有限の入力を持つ（部分）関数 $f : \mathbb{N}^k \to \mathbb{N}$ が Turing 機械によって**計算可能**であるとは，次のように動く Turing 機械 M が存在することをいう．

1. 各 $(i_1, \cdots, i_k) \in \mathbb{N}^k$ に対して，M の tape に $0^{i_1}10^{i_2}1\cdots10^{i_k}$ が入力として与えられる．
2. $f(i_1, \cdots, i_k)$ が定義されているときは，記号列 $0^{f(i_1,\cdots,i_k)}$ を tape の内容として停止する．
3. $f(i_1, \cdots, i_k)$ が定義されていないときは，停止しない．

注 3.2.2. 一般には Turing 機械 M ではなく，一般の算譜（定義 3.6.6）で良い．なお，未定義動作には全て停止させる Turing 機械の算譜がある（命題 3.1.6）から，定義としては，問題 $\mathbb{N} \to \mathbb{N}$ の未定義入力については，何を返しても，停止しなくても良い．

定義 3.2.3 ((total) recursive function, partial recursive function)**.** Turing 機械によって計算される関数を**帰納的関数**という．Turing 機械によって計算される部分関数を**部分帰納的関数**という．

3.3　Turing 機械の変種

off-line Turing machine　計算量理論で用いられるモデル．

counter machine　tape の代わりに counter を持ったモデル．オフライン Turing 機械のような入力テープと 2 つのカウンタがあれば Turing 機械と同等の計算能力をもつことが知られている．

pushdown automaton　2 つの stack で Turing 機械と同等の計算力を持つという点では counter 機械に一致する．

lambda calculus　LISP として実装されている．

random access machine　通常の計算機をモデル化したもの．アルゴリズムの時間やメモリの使用量はこのランダムアクセス機械で評価されることが多い．

parallel random access machine　並列コンピューティングに適用可能なアルゴリズムを設計するための抽象機械である．フリンの分類によれば，PRAM は MIMD 型コンピュータに相当する．

3.3.1　非決定性 Turing 機械

定義 3.3.1 (nondeterministic Turing machine)**.** 遷移関数を，遷移関係 $\delta \subset (Q \times \Gamma) \times (Q \times \Gamma \times \{L, R\})$ に変更したものを**非決定性 Turing 機械**という．

3.3.2　multitape Turing 機械

定義 3.3.2 (multitape Turing machine)**.** k 本の tape を持つ Turing 機械を **k-tape Turing 機械**という．

1. うち 1 本のみを入力用に使う．
2. 状態 q で各テープから $X_1, \cdots, X_k \in \Sigma$ を読んだとき，それに従って状態を $\mathrm{pr}_1(\delta(q, X_1, \cdots, X_k))$ に変え，$\mathrm{pr}_2(\delta(q, X_1, \cdots, X_k))$ を書き込み，各 head を $\mathrm{pr}_3(\delta(q, X_1, \cdots, X_k))$ 方向に移動する．従って，$\delta \subset (Q \times \Gamma^k) \times (Q \times \Gamma^k \times \{L, R\})$ である．

3.3.3　両方向に無限の長さを持つ tape

3.3.4　off-line Turing 機械

計算量理論で用いられるモデル.

定義 3.3.3 (off-line Turing machine). 入力 $x \in \Sigma^*$ は特殊な記号に挟まれた $¢x\$$ が印字された**入力テープ**で渡され，これは read-only である．これの各 cell を読むような書き込むことのできない，k-tape Turing 機械を**オフライン Turing 機械**という.

3.3.5　多次元 tape Turing 機械

両方向に無限の長さを持つ tape をさらに拡張して，n 次元の cell に分割された n 次元 tape を持った Turing 機械も考えられる.

3.3.6　線形拘束オートマトン

定義 3.3.4 (LBA: linear bounded automaton). tape の長さが有限で，特に入力文字列 $|w| = n$ に対して長さ $\exists k \in \mathbb{N}$, kn に制限される Turing 機械を**線形拘束オートマトン**という．この制限により LBA はある意味ではチューリングマシンよりも実在のコンピュータの正確なモデルと言える.

定義 3.3.5 (context-sensitive grammar). **文脈依存文法**とは，形式文法 $G = (V, T, P, S)$ であって，P の生成規則が次のように表せるものをいう：

1. $\alpha A \beta \to \alpha \gamma \beta \quad (A \in V, \alpha, \beta \in (V \cup \Sigma)^*, \gamma \in (V \cup \Sigma)^+)$.
2. $S \to \epsilon$.

注 3.3.6. この 2 つ目の規則を許すことで、空文字列を含む文脈依存言語の定義が可能になり、文脈依存言語が文脈自由言語を真に含むと言えるようになる。文脈依存言語の文法で制限されていることは、ある文字列から短い別の文字列へのマッピングを持たないことである.

命題 3.3.7 (文脈依存文法の特徴付け). 次の条件を満たす形式文法は，文脈依存文法である.

$$P = \{(\alpha, \beta) \in (V \cup \Sigma)^* \times (V \cup \Sigma)^* \mid |\alpha| \leqslant |\beta|\}.$$

注 3.3.8. この文法では生成規則を適用したときに文字列の長さが減ることがないので、「単調文法」(Monotonic Grammar) とか「非縮小文法」(Noncontracting Grammar) などと呼ばれる。非縮小文法と文脈依存文法は異なるが、同じ言語クラスを定義できるという意味でほぼ等価である（違いは、非縮小文法では空の文字列 ϵ を含む言語を生成できない点である）。文脈依存文法で記述された言語 L に対して、非縮小文法は $L \setminus \{\epsilon\}$ を記述できるし、その逆も真である。[†4]

定理 3.3.9. 言語 $L \subset \Sigma^*$ について，次の 2 条件は同値.

1. L は線形拘束オートマトンによって受理される.
2. L は文脈依存文法によって生成される.

文脈依存言語の決定問題

ある文字列 s が文脈依存文法 G で生成される言語に属するか否かという決定問題は、PSPACE 完全である。実際、文脈依存文法の中にはその文法認識問題が PSPACE 完全なものもある。[α]

[α] 文脈依存文法 (wikipedia)

[†4] 文脈依存文法 (wikipedia)

3.4　Church のテーゼ

こうした計算理論の結果を総括して，Alonzo Church (03-95) は次の提案をし，これは皆に理解れるている．この提案を Church のテーゼ (Church's Thesis) という：“計算可能”であるとは Turing 機械によって計算できることとしよう．

3.5　帰納的可算集合と帰納的集合

集合演算と，Turing 機械を素子とした論理回路が構成できるかどうかに同型対応を作っているように見える．分類理論 3.5.4 が成り立つ．
なお，帰納的関数は計算可能な全域関数 $\mathbb{N}^k \to \mathbb{N}$ のことで，帰納的可算な関数は部分関数 $\mathbb{N}^k \twoheadrightarrow \mathbb{N}$ のことである．前者の方が定義域には制限を受けているが，関数としては自由度が高い大きいクラスとなっている．層か？

命題 3.5.1 (補集合も帰納的)**.** 言語 $L \subset \Sigma^*$ が帰納的であるならば，その補集合 $\overline{L} = \Sigma^* \setminus L$ も帰納的である．

［証明］. 帰納的言語 L には，これを受理する停止する Turing 機械 M が存在して $L = L(M)$ となる．これに対して，M が yes を返すなら no を，M が no を返すなら yes を返すような停止する Turing 機械 M' が構成できる．これについて，$\overline{L} = L(M')$ である． □

命題 3.5.2 (合併も帰納的（可算）)**.** 言語 $L_1, L_2 \subset \Sigma^*$ について，

1. L_1, L_2 が帰納的ならば，$L_1 \cup L_2$ も帰納的である．
2. L_1, L_2 が帰納的可算ならば，$L_1 \cup L_2$ も帰納的可算である．

［証明］. M_1, M_2 をそれぞれ，L_1, L_2 を受理する（1. については停止する）Turing 機械とする．

1. M_1 が yes を返すならそのまま yes を，no を返すならば M_2 に同じ入力を入れて，yes を返すなら yes を，これでも no が返ってきたら no を出力する Turing 機械は，$L_1 \cup L_2$ を受理する．
2. M_1, M_2 に同時に入力を入れ，どちらかから yes が返ってきたら yes を出力する（他は不問）Turing 機械は，$L_1 \cup L_2$ を受理する．

□

命題 3.5.3. 言語 $L \subset \Sigma^*$ について，L も $\overline{L}$ も帰納的可算であるとき，L は帰納的である．

［証明］. L を受理する Turing 機械を M_1 とし，$\overline{L}$ を受理する Turing 機械を M_2 とする．全ての入力 $x \in \Sigma^*$ について $x \in L \vee x \in \overline{L}$ であるため，新たに Turing 機械 M を，M_1 から yes が返ってきたら yes，M_2 から yes が返ってきたら no を返すとすると，これは停止する Turing 機械で，$L = L(M)$． □

定理 3.5.4. 言語 $L \subset \Sigma^*$ について，次の 3 つのいずれかが成り立つ．

1. L も $\overline{L}$ も帰納的である．
2. $L, \overline{L}$ の一方が帰納的可算で，もう一方は帰納的可算でない．
3. L も $\overline{L}$ も帰納的可算でない．

所感 3.5.5. 停止性判定問題 3.6.11 で使う知見である．片方が帰納的であることを証明したら，その補集合も帰納的だと決まってしまうのが，停止する Turing 機械が受理する言語のクラス R の特徴である．

3.6　決定不能な問題

計算理論の本領発揮はここからである．文脈自由文法というメタ的な存在を研究対象として定義できたように，Turing 機械も研究対象とする．主な手法は還元算譜と標準形の理論である．
L_d は帰納的可算でなく $L_d \notin \mathrm{RE}$，L_u は $L_u \in \mathrm{RE}$ だが帰納的ではない：$L_u \notin \mathrm{R}$．従って，$\mathrm{R} \subsetneq \mathrm{RE} \subsetneq \{0,1\}^*$ がわかる．

3.6.1　Turing 機械の標準形

tape alphabet を $0,1$ にするとは，一つ一つの遷移を限りなく一口大にするということである．Assembly 言語に少し似ている．

補題 3.6.1 (tape alphabet の符号化)**.** 言語 $L \subset \{0,1\}^*$ が帰納的可算ならば，特に tape alphabet を，入力アルファベットと混用して $\{0,1,B\}$ とするような Turing 機械が存在してこれに受理される．さらに，この Turing 機械は $|F| = 1$ で，その受理状態で停止するとして良い．

［証明］. 命題 3.1.6 より，受理することと停止することは等価にできる算譜がある．その受理状態を 1 つにまとめる算譜と，一般の Turing 機械から tape alphabet を $\{0,1,B\}$ に還元する符号化の算譜が必要である．　□

3.6.2　Turing 機械の符号化

Turing 機械の符号化にあたっては，0 は個数に意味を持たせることで自然数上の意味論を tape 上に載せ，1 は区切りとして使う．1 の個数に区切りとしての意味論を載せる．この形式文法（？）の意味論の射程は極めて広いようだ．

定義 3.6.2 (encoding of Turing machine)**.**

1. 補題 3.6.1 の標準形に還元した Turing 機械は，一般に
$$M = (Q = \{q_1, \cdots, q_n\}, \{0,1\}, \{0,1,B\}, \delta, q_1, B, \{q_2\})$$
と表せる．
2. 記号 $0,1,B$ をそれぞれ X_1, X_2, X_3 とし，head の動く方向 L,R は D_1, D_2 とする．
3. 動作（ID 間の二項関係）$\delta(q_i, X_j) = (q_k, X_l, D_m)$ は，記号列 $0^i10^j10^k10^l10^m$ に符号化する．
4. Turing 機械 M の符号化は，M の動作の符号化 $\{\mathrm{code}_i\}_{i\in[r]}$ を用いて，$111\mathrm{code}_1 11\mathrm{code}_2 11 \cdots 11\mathrm{code}_r 111$ とする．
5. 符号化全体を CODE とし，Turing 機械全体を TM とすると，今定めた符号化は写像 CODE→TM を定め，これは単射ではない．Turing 機械の符号化は動作の table $(\mathrm{code}_i)_{i\in[r]}$ の順番に関して自由度が残るからである．
6. Turing 機械 M と入力 $w \in \{0,1\}^*$ について，組 (M,w) の符号化 $\langle M,w\rangle \in \{0,1\}^*$ を M の符号化の後に w を並べて得られる文字列とする．

例 3.6.3. Turing 機械 $M = (Q = \{q_1, \cdots, q_3\}, \{0,1\}, \{0,1,B\}, \delta, q_1, B, \{q_2\})$ で，
$$\delta = \left\{ \begin{array}{ll} ((q_1,1),(q_3,0,R)), & ((q_3,0),(q_1,1,R)), \\ ((q_3,1),(q_2,0,R)), & ((q_3,B),(q_3,1,L)) \end{array} \right\}$$
とする．このとき，M の符号化は
$$1110^110^210^310^110^2110^310^110^110^210^2110^310^210^210^110^2110^310^310^310^210^1111$$

命題 3.6.4. 言語 $L = \{y \in \{0,1\}^* \mid y \text{ は或る Turing機械の符号化となっている}\}$ は帰納的集合である．即ち，任意の記号列 $y \in \{0,1\}^*$ に対して，これを符号化とする Turing 機械が存在するかどうかを判定する停止する Turing 機械 M_{form} が存在する．

［証明］.　□

3.6.3　枠組み

こうして計算理論の枠組みが整備される．0, 1 は必ずしも扱いやすくはないので，同型 $(w_i) : \mathbb{N} \to \{0,1\}^*$ を取り，自然数論の上に構築される．

定義 3.6.5 (problem)**.** 部分関数 $\mathbb{N} \to \mathbb{N}$，あるいはそれを定めるメタ言語の文字列を**問題**という．未定義のものがあって良い，これは Turing 機械が停止しないことに対応する．また，多価関数（関係）として定義しても良い．

定義 3.6.6 (algorithm)**.** 処理の手順を適切な言語で定めたものを**算譜**という．特に，遷移（部分）関数 $\delta \subset ((Q \times \Gamma) \times (Q \times \Gamma \times \{L, R\}))$ を**算譜**という．勿論多価関数（関係）として定義しても良い．

注 3.6.7. Church のテーゼは，算譜と機械を同義に扱っているというのが興味深い．これが計算理論の摩訶不思議な雰囲気の始まりである．計算理論は物理学とどう関わってくるのか．

3.6.4　受理の認識機械の非存在

定理 3.6.8 (受理の認識機械の非存在)**.**

$$L_d = \left\{ w_i\{0,1\}^* \;\middle|\; \begin{array}{l} w_i \text{は或る Turing機械 } M_{w_i} \text{の符号化であり，} \\ \text{それは文字列 } w_i \text{を受理する：} w_i \in L(M_{w_i}) \end{array} \right\}$$

は帰納的可算集合でない．

［証明］.

表の定義　$\{0,1\}^*$ の要素の列 $(w_i)_{i\in\mathbb{N}} : \mathbb{N} \to \{0,1\}^*$ を

$$\epsilon, 0, 1, 00, 01, 11, 000, 001, 010, 011, 100, 101, 110, 111, \cdots$$

とする．[†5] i 番目の要素 w_i を符号化とする Turing 機械を，存在するならば M_{w_i} と名付ける．これを用いて，表 $(a_{ij})_{i,j\in\mathbb{N}} : \mathbb{N}^2 \to \{0,1\}$ を次のように定める．

$$a_{ij} \begin{cases} a_{ij} = 0, & M_{w_i} \text{が存在して，} w_j \text{を受理しないとき：} w_j \notin L(M_{w_i}), \\ a_{ij} = 1, & M_{w_i} \text{が存在しない，または存在して } w_j \text{を受理するとき：} w_j \in L(M_{w_i}). \end{cases}$$

この表は命題 3.6.4 より計算可能である．

構成　ここで，$L_d := \{w_i \in \{0,1\}^* \mid i \text{ は } a_{ii} = 0 \text{ を満たす}\}$ とする．即ち，$w_i \notin L(M_{w_i})$ を満たす記号列 w_i を集めたものである．この言語 L_d を受理する Turing 機械が存在すると仮定すると，$\exists i \in \mathbb{N}$, $L(M_{w_0}) = L_d$ である．ここで，Turing 機械の符号化 w_i 自身を入力と見ると，$w_i \in L(M_{w_i}) \vee w_i \notin L(M_{w_i})$ が成り立つ．$w_i \in L(M_{w_i})$ の時，$a_{ii} = 1$ であるから，$w_i \notin L_d$ である．これは $L(M_{w_0}) = L_d$ に矛盾．従って $w_i \notin L(M_{w_i})$ であるが，この時 $a_{ii} = 0$ であるから，$w_i \in L_d$ であり，これも $L(M_{w_0}) = L_d$ に矛盾．よって，言語 L_d を受理する Turing 機械は存在しない．

□

所感 3.6.9 (有限という本質的障壁？)**.**

1. Turing 機械を符号化 $\{0,1\}^* \supsetneq$CODE$\to$TM を通じて，$(w_i)_{i\in\mathbb{N}} : \mathbb{N} \to \{0,1\}^*$ を用いて自然数 $\mathbb{N}$ に引き戻している．そう，抽象機械の variation は本質的に可算個なのである．一方で問題全体の集合＝言語の集合は $P(\{0,1\}^*)$ だけある．これは連続体濃度である．ここに本質的な限界がある．一つ一つの機械は有限なので，何かしらの方法で無限を創発せねばならぬ．
2. この対角線論法は，Turing 機械の符号化と，問題文に同じアルファベット $\{0,1\}$ を用いて同じ俎上に載せているからこそできる議論である．これが最大のトリックなのではないか．むしろ，このような証明が存在してしまうということが壮大な背理法なのである．

[†5] ${}^{\omega>}\{0,1\} \simeq \{\epsilon, 0, 1\}^{\mathbb{N}}$ より，これは全射ではないはずである．かと言ってこの列に含まれない記号列を挙げるアイデアもない．いや，$\{\epsilon, 0, 1\}^{\mathbb{N}}$ の有限な部分集合に限れば全射か？無限長のものは生成しないのか．じゃあ，${}^{\omega>}\{0,1\} \simeq \{\epsilon, 0, 1\}^{\mathbb{N}}$ が間違っているのか．正しくは ${}^{\omega>}\{0,1\} \hookrightarrow \{\epsilon, 0, 1\}^{\mathbb{N}}$ か．

3. 言語の分類定理 3.5.4 によると，$\overline{L_d}$ は帰納的でないことはわかるが，帰納的可算であるかどうかは定まらない．

3.6.5　万能 Turing 機械

ここで初めて機械と算譜の概念が分離する．なぜだろう．機械の方がメタ的な概念であったから，物理学と数学の関係性はあるのであろう．また，object というものがプログラムに優先するようになった起源でもあるかもしれない．プログラム自体が入力として引き渡せるというのは現代では自然な感覚であるが，この議論は Turing が 36 年に作ったものである．
そしてこのメタ機械＝万能 Turing 機械についてのメタ定理を得ることになる：停止性判定問題は解けない．一つ下の次元では，$L_u \notin R$ を意味する．即ち，L_u を解く Turing 機械＝万能 Turing 機械は，yes の時はいずれ教えてくれるが，no の時は停止するかも予測がつかない．これが，Turing 機械の有限性と，その総数の可算性と，言語の連続体濃度とが編み上げている模様である．

定理 3.6.10 (万能 Turing 機械の構成)．言語 $L_u := \{\langle M, w\rangle \in \{0,1\} \mid \text{Turing機械 } M \text{ は } w \text{ を受理する }\}$ は帰納的可算である．即ち，L_u を受理する Turing 機械 M_1 が存在する．

［証明］．3 本の tape を持った multitape Turing 機械 M_1 を構成する．第一の tape に入力記号列 $\langle M, w\rangle$ を格納し，第二のテープを M のテープを模倣するために使い，第三のテープを M の状態を code するために使う．M_1 の算譜 $\delta \subset (Q\times\Gamma)\times(Q\times\Gamma\times\{L,R\})$ を次のように設定する．

1. 第一 tape の内容が，動作 $\delta(q_i, X_j) = (q_k, X_l, D_m)$ は，記号列 $0^i10^j10^k10^l10^m$ に符号化する定義 3.6.2 の文法にあっているか確認する．また，同じ接頭語 0^i10^j をもち，違う接尾語を持つ code が存在しないことを確認する．また，$1 \leqslant j, l \leqslant 3$ と $1 \leqslant m \leqslant 2$ も確認する．この時第三の tape を使う．
2. 入力文字列 w の内容を第二 tape にコピーし，第三 tape は初期化して初期状態 q_1 の符号化 0 を書き込む．
3. 第三 tape の内容が 00 となると，これは q_2 の符号化であるから，受理として M_1 を停止させることとする．
4. 第二 tape の head が記号 X_j を読んでいて，第三 tape の内容が 0^i である時，第一 tape の内容の 111 で囲まれた領域であって 0^i10^j1 で始まる code を探す．この code がなければ受理せず停止することとする．この code があり，code の全体が $0^i10^j10^k10^l10^m$ であるとしたら，第三 tape を 0^k に書き直し，第二 tape に X_l を書き込み，m に従って head を移動させる．step 3 に戻る．

□

定理 3.6.11 (停止性判定問題は解けない)．言語 L_u は帰納的ではない．

［証明］．L_u を受理する停止する Turing 機械 M が存在すると仮定して，矛盾を示す．この万能 Turing 機械 M を用いて構成した次の停止する Turing 機械 M_2 は，言語

$$\overline{L_d} = \left\{ w_i \in \{0,1\}^* \;\middle|\; \begin{array}{l} w_i \text{は Turing機械の符号化ではない，または} \\ w_i \text{は Turing機械の符号化でこれを受理する：} w_i \in M_{w_i}. \end{array} \right\}$$

を受理する．

1. M_2 はまず $w_i \in \{0,1\}^*$ が Turing 機械の符号化であるかどうかを確かめ，そうでないならば受理して停止する．これは構成できる．
2. 次に，Turing 機械の符号化だった場合，M に w_iw_i を入力することで，これが受理される時とは $w_i \in M_{w_i}$ である場合に他ならない．

従って，$\overline{L_d}$ は帰納的であるため，言語の分類定理 3.5.4（命題 3.5.1）より，L_d も帰納的である．が，これは定理 3.6.8 より，L_d は帰納的可算でさえないことに矛盾．従って，M は存在しない．□

所感 3.6.12. 「理解してきた，計算不能は有限性と関係があるのか．例えば「解けない時には解けないといえ」と言われると，定義域が広がり過ぎて（対応させなきゃいけない定義域内の元が多過ぎて，原始再帰的には定義できない），有限の範囲では答えを返す算譜がないって感じだろうな．ああ，だから計算不可能性が再帰関数と関係があるのかもしれない．これは要は関数の定義可能

性じゃないか.」という昔のメモ書きがある.

3.6.6　その他の決定問題

決定問題とは部分関数 $\{0,1\}^* \nrightarrow \{0,1\} = \mathrm{TV}$ で，有限列の符号化を用いて $\mathbb{N}$ の内容をさらに還元したものであり，高度に形式科学的な対象である.

半決定可能な問題＝ $\mathrm{RE} \setminus \mathrm{R}$

- Turing 機械の停止性問題（定理 3.6.11）
- 受理計算（万能 Turing 機械，定理 3.6.10）
- 関係 $\mathrm{L}(\mathrm{M}) = \emptyset$
- 文脈自由文法 G に対して，関係 $\mathrm{L}(\mathrm{G}) = \Sigma^*$．上と共役な問題だろうな.

半決定可能でない問題＝ $\overline{\mathrm{RE}}$

証明には，「この問題が解けると，停止性問題が解けてしまう」という還元の手法を用いる.

- 半決定可能な問題の補集合（命題 3.5.1 より）
- 関係 $\mathrm{L}(\mathrm{M}) \in \mathrm{R}$．停止性問題に還元できそう.
- 関係 $\mathrm{L}(\mathrm{M}_1) = \mathrm{L}(\mathrm{M}_2)$.

3.7　計算可能解析学

そうか，物理量は有理数なのではなくて，「計算可能実数」「測定可能実数」なのかも知れない.

定義 3.7.1 (computable reals)**.** 実数であって，有限かつ停止する算譜によって，任意精度で計算できるようなものを**計算可能実数**という.

命題 3.7.2. 計算可能実数の全体は実閉体を成す（Weihrauch 2000, p. 180）.

計算可能実関数は計算可能実数を計算可能実数に写す。計算可能実関数の合成関数は再び計算可能となる。任意の計算可能実関数は連続である（Weihrauch 2000, p. 6）。リーマン積分は計算可能な作用素である：換言すれば、任意の計算可能関数について、その積分を数値的に評価するアルゴリズムがある、一様ノルムを取る演算もまた計算可能である。これがリーマン積分の計算可能性を導く。実数値関数の微分作用素は計算不可能であるが、複素関数に対するそれは計算可能である。後者の結果はコーシーの積分公式および積分の計算可能性から従う。前者の否定的結果は（実数値関数上の）微分が不連続であるという事実による。これは、実解析と複素解析の間の隔たりを示している。また、しばしば前述の積分公式や自動微分によってバイパスされる、数値微分の困難さも示している。[†6]

[†6] 計算可能解析学 (wikipedia)

参考文献

[1] Pullum, Geoffrey K.; Gerald Gazdar (1982 年). “Natural languages and context-free languages”. Linguistics and Philosophy 4: 471 - 504.

[2] Shieber, Stuart (1985 年). “Evidence against the context-freeness of natural language”. Linguistics and Philosophy 8: 333 - 343.

[3] Culy, Christopher (1985). “The Complexity of the Vocabulary of Bambara”. Linguistics and Philosophy 8: 345 - 351.

[4] Mealy, George H. (1955), "A method for synthesizing sequential circuits" (PDF), Bell System Technical Journal, 34: 1045 - 1079, doi:10.1002/j.1538-7305.1955.tb03788.x, MR 0073450.

[5] Edward Moore. "Gedanken-experiments on Sequential Machines". pp. 129 - 153, Automata Studies, Annals of Mathematical Studies, no. 34, Princeton University Press, Princeton, N. J., 1956.

[6] Y. Bar-Hillel, M. Perles, E. Shamir, "On formal properties of simple phrase structure grammars", Zeitschrift für Phonetik, Sprachweissenshaft und Kommunikationsforschung 14 (1961) pp. 143-172.