オートマトン・計算と計算理論

岩間一雄（京都大学）

まえがき

ちなみに手元の計算機からアマゾンを呼んで，オートマトンや言語理論で検索してみると，20 冊程度の本が現れる．つまり，かなりの本が既に執筆されているということである．この期に及んで，更に新たな一冊を投入するとなればかなりの覚悟が要求されるであろう．屋上屋を架すことにならないよう注意したいものである．そこで，まず，本書をどのような方針で執筆するかをできるだけ丁寧に説明させていただく．

オートマトン・言語理論は情報関係の学科の典型的な基礎科目である．従って，かなりの数の学生諸君が勉強されるわけであるが，その中で，将来専門家として研究を志す人はほとんど皆無であろう．また，直接何に役立つのかもはっきりしない．同じ基礎科目でも，例えばアルゴリズムとデータ構造などは，プログラムを書くときにすぐ役に立つし，面白くなってしまって研究者の道に入る人もかなり存在する．このように，後者は勉強するモーティベーションが分かりやすい．（勉強するために一番重要なのは言うまでもなくモーティベーションである．あのつまらない大学入試の勉強ができたのも大学進学という如何ともしがたいモーティベーションの賜物である．）

それではオートマトン・言語理論を勉強するとどんな利益があるのであろうか．一言で言ってしまえば，「計算機に負けない体をつくれる」ことである．（多少年輩の読者の方であれば，昔冬になると，「寒さに負けない体をつくれ」と小学校の先生が口癖の様に言っておられたことを思い出すであろう．）計算機は多分最も複雑な機械の一つであろうが，その原理は驚くほど簡単である．その簡単な原理を理解してしまえば，もう，計算機に関する様々な困難に遭遇しても，「何とかなる」という不思議な自信が湧いてくるのである．品質の高い解説なら，基礎から一歩一歩積み上げても，それほど時間を費やすこと無く比較的高度な知識が身につくという学習の方法論のようなものも体得できる．従って，例えば，パソコンの OS を勉強する必要が生じた時にも，いたずらに平積みの高さに惑わされることなく，上記の方法論にそった良質な解説書を見抜く眼力を養うこともできるのである．逆に言えば，このようにして眼力を養った読者の方々にとっては，適当に数ページを眺めてみて理解できないようであれば良質の解説書ではないと判断する妙な自信のようなものも得られる．

そこで，本書では，計算機の原理あるいは計算機による計算の原理が理解できることを最大の目的とした．その目的に最短距離で到達できるように，いわゆる知識としての内容は最小限に絞った．例えば，コンパイラの設計に言語理論は必須であるが，上記の目的とは若干異なる．従って，本書ではコンパイラ設計にとって重要な内容は，再び基本方針にそってその原理的なところには触れるが，大部分を削除して他の解説書に委ねることにした．記述の方法にしても，できるだけ形式的記述を避け，直観に訴える記述を採用した．従って，理論書に付き物の様に言われる苦難の努力なしで「解った気」になってもらえるように（筆者自信が苦難の）努力と今までの経験を最大限つぎ込んだつもりである．（直観に訴える記述は危険性が高いと言われている．とんでもない誤解がしばしば生じるからである．しかし，筆者はそのようは誤解は，勿論避ける努力を随所に盛り込んではいるが，勉強が続かなくなってしまうよりはずっと優だと考えている．）

大学の専門過程の半期の講義に使用して頂くことを考えている．しかし，事前の知識は全くと言って良いほど要求しないので，カリキュラムの編成によっては入学直後の学生諸君でも全く問題ないし，高校生の諸君でも意欲さえあれば最後まで読み進んで頂けるものと信じるし，勿論，文系・理系は問わない．なお，本書をどのように利用して頂くのが良いかについては，関連の講義科目にも触れてより突っ込んだ解説（筆者の個人的見解以上のものではないとのお叱りをうけるかもしれないが）を 1 章の終りの節にもうけた． 1 章の導入部を読んで頂いた後の方が多少でも理解してもらいやすいと考えたからである．

本書のテーマに関しては筆者は十分な教育経験を積んでいると自負している． 20 年以上に渡って，京都産業大学，九州大学，京都大学で講義を担当しているし，この間，カルフォルニア大学バークレー校でも講義を担当し，詳細な講義ノートを作成した．この英文の講義ノートが本書の元になっていると言って良い．初学者の陥り易い間違いも十分に分かっているつもりである．従って，随所に意識的に間違った証明を与え，それを修正するという形の記述を取り入れた．結構頻繁に取り入れたので，「またか！」と辟易される読者の方がおられるかも知れないので，ここであらかじめ謝罪させて頂く．種明かしをする前にこの様に見破っていただければ，筆者としては望外の幸せである．

本書の執筆に当っては多くの方にお世話になった．．．．

第 1 章

言語とは何か・なぜ必要か

言語とは何か，なぜ必要かというのは（言語学の専門家の方に取っては大問題かもしれないが，少なくとも普通の人にとっては）幼稚な質問である．人と人がコミュニケーションするのに利用する道具である．人と人に限らず，人と計算機が会話する場合にも言語は利用される．例えば，プログラミング言語と呼ばれているものがそうである．計算機と計算機が会話するための言語も数多く存在する．

本書の最初の章の目的は我々がこれから議論する形式言語という数学的モデル（モデルという言葉に違和感を感じる方は模型飛行機を思い浮かべて頂けば良い．現実の複雑なものを単純化・抽象化した「もの」である）をできるだけ自然に導入することである．そこで，まず，文章の意味を理解したり，文章が文法的に正しいかどうかを判定するといった言語の処理を，例題を通して理解することによって言語に対するイメージを養ってもらう．そのあとで，自然な延長として，我々のモデルである形式言語を導入する．次章以降ではこの言語のモデルを計算機のモデルで処理するという過程を通して計算機の原理を理解して頂く．これが通常「オートマトンと言語理論」と呼ばれる学問なのである．

1.1 文章の意味を理解する

人と人が会話する場合はそれぞれが瞬時に意味を理解することができるのであるが，この「意味を理解する」という作業を計算機に実行させようとすると結構やっかいである．異なった言語間（例えば英語と日本語）の機械翻訳は計算機が誕生した時からの最も自然で有名な計算機の利用例であるが，ごく最近までまともなシステムは存在しなかった．人と計算機，計算機と計算機の会話に利用する言語では，この意味を理解する作業をどうしても計算機にさせる必要がでてくる．この節では意味を理解するとは一体どのような作業なのかを考えてみたい．

以下の様な数式の計算をしたいとする．そのためにはその（文章の）意味を理解する必要がある．

$$23 + ((15 + 30)/(7 + 2 \cdot 4) + 51) \cdot (8 + 13).$$

数式が文章であるというのはちょっと乱暴であるが，まあ辛抱して欲しい．さて，この数式の「意味」とはそもそもいかなるものでどんな表現が可能なのであろうか．ここでは以下のように

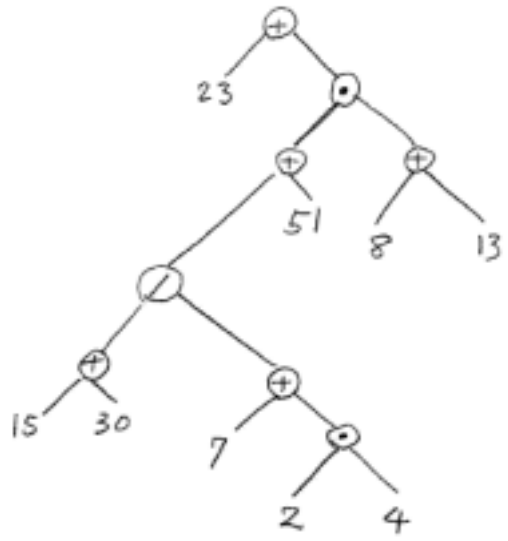

図 1.1 数式計算の木

考えよう．2 つの数の加減乗除しか知らない小学生にこの式の計算の仕方を教える場面を考えてほしい．もし，何らかの分かりやすい記述（説明）で上手に教えることができたなら，その記述を上の式の意味ととらえても良いであろう．

ここでは，この様な記述として，図 1.1 に示すような**木**を導入する．木とは，グラフの一種であり，実際の植物の木を逆さにした様な構造を持っている．木は**頂点**と**枝**からなる．一番上の頂点を木の**根**と呼ぶ．一番下の頂点（今の場合は 23， 15， 30 などの数値が書いてある）を**葉**と呼ぶ．一番下というのを 2 と 4 の葉だけと考えないで欲しい．つまり，それより更に下に頂点が無いという意味である．葉以外の頂点には $+$ や $\cdot$ 等の演算記号が付随している．また，葉以外の頂点 v はその下に 2 つの頂点を有している（例えば根の頂点の下には 23 と $\cdot$ の頂点）がそれらを v の**子**と呼び， v 自身はそれら 2 つの子の**親**と呼ぶ．子の子（更にまたその子）を**子孫**と呼んだりする．**先祖**もまた同様である．

さて，小学生への説明は以下の様に行う．木の頂点を見ていって，もしある頂点 v の 2 つの子供の頂点の値が共に決定されていたら v の値を決定する．決定の方法は，単に頂点 v に付随している演算を 2 つの子供の頂点の値に適用するだけである．図 1.1 を見ていくと，最初は例えば根を見ると，その左の子の値は決定（23）されているが，右の子の値が決定されていない．従って，下の方に進んで行くと（その進み方は適当で構わない），例えば右に右に進んで行って，8 と 13 の親に来る．この場合その親の値を 21 に決定する．このような作業を進めて行けば，いずれ根の頂点の値が決定される．その値が式の値である．各頂点の値を決めるのに必要な知識は 2 つの数の演算のみであるであることに注意されたい．つまり，我々の意図は，上の式の意味として図 1.1 の木を対応させようということである．意味そのものというのはちょっと苦しいかもしれないが，少なくとも意味を理解する上で大きな助けになっている．ここでは小学生を使ったが，この小学生を計算機に置き換えて見よう．図 1.1 の木が与えられたのであれば上の方法で根の値を決定するようなプログラムを書くことはそれほど困難ではないであろう．しかし，式から直接値を計算しようとすれば，プログラムは結構大変になることを感じ取って欲しい．つまり，式から木に直すことが言語の処理で，それが本質的部分だと言いたいのである．

優秀な読者諸君にとっては，このような木構造を文章に対応させることは初めての経験ではないであろう．例えば英語の文法の勉強の時に，文章は主語と述語から成っていて，述語は動詞と副詞句から成って，といったルールを覚えているであろう．この場合もルールを適用すると文章から自然に木が導かれる．この木がその文章の意味を理解するのに大きな助けになることは上の数式の場合と同様である．このようなルールを**文法**と読んでいる．文法に従って作成された木を**導出木**と呼ぶ．導出木は，我々の形式言語の議論の中でも中心的役割を演じる．

1.2　文章の正しさを解析する

プログラムを書いても，度重なるエラーで悩まされた経験をお持ちの読者の方も多いと思われる．前節で言った様な文章の意味が定まるためにはその文章が「正しい」ものであることが条件になる．それでは正しい文章とはいかなる文章なのであろうか．

再び数式を例にとって見てみよう．数式からカッコのみを残して他の数値や演算子を消した後で以下の様なカッコの並びが得られたとしよう．

$$(()(())(()).$$

これは正しいカッコの並びではない．なぜなら，左カッコの数が右カッコの数より多いからである．それでは次のカッコの並びはどうであろうか．

$$(()((()))))(()(())$$

この場合，左カッコの数と右カッコの数は同一であるが，やはりよく見ると何かおかしい．このように式からカッコだけを取り出した文章をカッコ文と呼ぶことにしよう．

練習問題 1.1　カッコ文の正しさを検証する方法を与えよ．

解答　次のような方法はいかがであろうか．左から各カッコの下に整数の数値を書いていく．初期値は 0 である．左カッコに出会ったら，その一つ左側のカッコの下の数値（存在しない場合は初期値）に 1 を加えて現在の左カッコの下の数値とする．右カッコの場合は 1 を引くことを除いて同じである．上の 2 つ目の例で言うと，最初の（一番左の）カッコの下が 1，次が 2，次が 1，という様に続いていく．最後まで数値がついたときに，負の数値が存在しないで，且つ最後の（つまり一番右の）カッコの下の数値が 0 になっていたら正しいカッコ文と判断する．プログラムになれている方なら，スタックの利用を思いつかれるかも知れない，つまり，左の端から見ていって，左カッコはスタックに積み，右カッコはスタックから 1 個の左カッコを取り出す．この動作が最後まで続いて（つまり，取り出すべき左カッコが常にスタックに存在し），最後のカッコを処理した後でスタックが空になっていれば正しいカッコ文である．（解答終り）

この様に与えられた文章が正しいものかどうかを判定するための計算機のモデルとして，我々は**オートマトン**と呼ばれる（機械の様な）ものを導入する．前述の文法と並んで我々の議論の中心を成すものである．

1.3　形式言語を導入する

以上の議論で「言語」や「文章」の大雑把なイメージがつかめたと思う．このイメージを損うこと無く，数学的議論に敵した簡潔さと厳密さを有したモデルを導入しよう．モデルの構成要素は，より基本的なものから上位に向かって，文字，文章，言語から成る．

まず，最も基本になるのが，**文字**である．これは**記号**とも呼ばれ，我々の場合は後者を多用する．自然言語の場合は，日本語なら，文字としてはひらがな，カタカナ，漢字等を利用する．形式言語の場合は，$0, 1, a, b, c, \cdots$ の様な記号を多用する．記号の有限集合を**アルファベット**と呼ぶ．形式言語でよく使用されるアルファベットとしては，$\{0, 1\}$ や $\{a, b, c\}$ 等がある．勿論，自然言語の場合はより多くの文字を使用する．例えば，英語の場合は数十の文字を使用するし，日本語の場合はそれより遥かに多い．しかし，重要なことはアルファベットは常に有限集合であることである．

アルファベット Σ（ギリシャ文字については次の節を参照されたい）上の**文章**とは，Σ の記号を重複を許して並べたものである．例えば，$\Sigma = \{0, 1\}$ なら，01101 等が Σ 上の文章である．文章は**記号列**や単に**列**とも呼ばれる．我々は最後の列を多用する．列に含まれる記号の個数をその列の**長さ**と呼ぶ．例えば列 $x = 01101$ の長さは 5 である．列 x の長さを $|x|$ で表すこともある．長さが 0 の列も使えると便利である．我々はアルファベットに関わらず，長さ 0 の列を常に ϵ で表す．（記号が 0 個なので陽に書けない．そこで，特別な記号 ϵ を使用するのである．なお，負の長さの列は存在しない．）列 x の**逆**とは x を後ろから逆に読んだ列のことで，x^{R} で表す．例えば，$x = 1101$ の時，$x^{\mathrm{R}} = 1011$ である．2 つの列，x, y に対して，x と y の**連接**とは単に列 x の後ろに y をくっつけた列のことで xy と書く．ただし，連接という演算を強調して $x \cdot y$ と書く場合もある．例えば $x = 1011$，$y = 000$ なら，$xy = 1011000$ である．同じ列の連接は，xx を x^2 の様に書いてもよい．3 つ以上の列の連接も同様で，xyz の様に書いてよい．（正確に言えば，連接はあくまで 2 項演算であるから，$(xy)z$ 又は $x(yz)$ と書くのが正しい．ただ，連接は明らかに結合律をみたすので，xyz と書いても混乱は起こらない）．いかなる列 x に対しても $x\epsilon = \epsilon x = x$ である．これは，ϵ が長さ 0 の列であることから当然であろう．列 x の一部分の列を**部分列**という．例えば，ab や bba は列 $bbabb$ の部分列である．正確に言うと列 w が（ϵ でもよい）列 x, y, z に対して $w = xyz$ と書けるとき，列 y を w の部分列と呼ぶ．

さて，いよいよ言語の定義である．アルファベット Σ 上の**言語**とは，Σ 上の列の集合である．これだけではどんな集合なのか分からないが，どんな集合でもよいのである．例えば，$\Sigma = \{0, 1\}$ の時，有限集合 $\{0, 01, 000, 10101\}$ は立派な言語である．しかし，我々は通常ある性質を持った列の集合を考えることが多い．例えば，

$$\{0^n 1^n \mid n \geq 0\}$$

は前半部が 0，後半部が 1 でそれらの個数が正確に等しい列の全体からなる集合（つまり言語）である．n はいくら大きくてもよい整数なので，この言語は無限集合である．以下言語といえば通常無限集合である．以前の例の様に，有限集合は単純に書き下してしまうことができるので興味が少ない．無限集合はもはや書き下すことはできないので，それをいかにして表現するかが我々の主題なのである．なお，上の様な言語の表現方法に慣れて欲しい．$\{A \mid B\}$ で，A の部分に列の一般形を書き，B の部分にその一般形が集合に入るために満たすべき条件を書く．上の

例で 0^n1^n は 0 を n 個と 1 を n 個並べたものの連接を表しており，B でそれらの個数が 0 個以上何個でもよいことを示している．本書で（上の n の様な）数が出てきたらそれはほとんどの場合非負の整数であり，負の整数や実数などの場合は必ずその旨を明記する．

特別な言語として，アルファベット Σ 上の全ての列の集合（言語の条件を明らかに満たしている）を Σ^* で表す．例えば $\{0,1\}^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, \cdots\}$ であり，当然無限集合である．この逆で，どんな列も含まない，空集合もまた言語である．本書では空集合は常に ϕ で表す．長さ 0 の列（これを以後**空列**と呼ぶこともある）ϵ と空集合 ϕ をくれぐれも混同しないで欲しい．ϵ だけからなる言語 $\{\epsilon\}$ は（一つの要素を含んでいるので）空集合ではない．

練習問題 1.2　列 x の逆をより形式的に定義せよ．

解答　再帰的定義という手法を知っておくと便利である．アルファベット Σ 上の列 x（簡単に $x \in \Sigma^*$ と書くこともある）の逆 x^{R} は以下の (i) と (ii) で再帰的に定義される．(i) $\epsilon^{\mathrm{R}} = \epsilon$, (ii) 列 $x \in \Sigma^*$ と記号 $a \in \Sigma$ に対して，$(ax)^{\mathrm{R}} = x^{\mathrm{R}}a$．（注意深い読者の方は (ii) で記号 a と列 x の連接 ax が未定義であることに気が付くであろう．これは，a を記号と考えずに長さ 1 の列と考えることで解決する．）例えば，$(abc)^{\mathrm{R}}$ は以下のように cba であることが導かれる．

$$(abc)^{\mathrm{R}} = (a(bc))^{\mathrm{R}} = (bc)^{\mathrm{R}}\cdot a = (b(c))^{\mathrm{R}}\cdot a = (c)^{\mathrm{R}}\cdot b\cdot a = (c(\epsilon))^{\mathrm{R}}\cdot b\cdot a = \epsilon^{\mathrm{R}}\cdot c\cdot b\cdot a = \epsilon\cdot c\cdot b\cdot a = cba$$

上で与えた (i) と (ii)，さらに他の定義をどのように適用したかを細部まで吟味して頂きたい．（解答終わり）

練習問題 1.3　x と y を列としたとき，$(xy)^R = y^R x^R$ であることを証明せよ．

解答　本問題を通して数学的帰納法を復習しておこう．列 xy の長さに関する帰納法で証明する．(i) まず $|xy| = 0$ の時に証明すべき内容（これをしばしば命題と呼ぶ）が正しいことを言う．$|xy| = 0$ なら $x = y = \epsilon$ しかあり得ないから，$(\epsilon\epsilon)^{\mathrm{R}} = \epsilon^{\mathrm{R}} = \epsilon$．また，$\epsilon^{\mathrm{R}} \cdot \epsilon^{\mathrm{R}} = \epsilon\epsilon = \epsilon$ であり，左辺と右辺が等しいので命題は成立する．(ii) $|xy| = n$（$n \geq 0$）の時命題が正しいと仮定して $|xy| = n+1$ の時も命題が正しいことを示そう．まず，$x = \epsilon$ の時は簡単なので読者にお任せする．$x \neq \epsilon$ とする．x の最初の記号を a としよう．すると，$x = ax'$ と書けるので，命題の左辺は

$$(xy)^{\mathrm{R}} = (ax'y)^{\mathrm{R}} = (a(x'y))^{\mathrm{R}} = (x'y)^{\mathrm{R}} \cdot a = y^{\mathrm{R}}x'^{\mathrm{R}} \cdot a.$$

ここで，最後の変換に帰納法の仮定（つまり，$x'y$ の長さが n なので $(x'y)^{\mathrm{R}} = y^{\mathrm{R}}x'^{\mathrm{R}}$ が成立）を使っていることに注意せよ．一方，命題の右辺は

$$y^{\mathrm{R}}x^{\mathrm{R}} = y^{\mathrm{R}}(ax')^{\mathrm{R}} = y^{\mathrm{R}}x'^{\mathrm{R}} \cdot a.$$

となって，命題が成立することが分かった．（解答終わり）

ここで一休みして，現実の言語との対応を考えておこう．日本語という言語は，日本語で使用できる文字からなる文章の集合で，文法的に正しいものと定義するのが妥当であろう．ここで，「文章」は例えば，この段落の最初の一文「一休みして．．．．考えておこう．」を文章と考えるのが普通であるが，そうすると今度はこの段落全体，あるいはもっと大きく，この本全体は何なのかという疑問が出てくる．そこで，文章をもっと拡大して考えて，ピリオドで終わる通常の文章を連接したものもまた文章であると考える．そうすれば，この本全体も一つの文章になって

この疑問も解消する．上の無限集合のところでも見たように文章はいくら長くてもよいのである．こうして，日本語もまた，言語，文章，文字の 3 レベルの枠組みに納まる．

さて次に，色々と便利なので，言語と言語の**連接**を（勿論，列と列の連接の自然な延長として）定義しよう． L_1 と L_2 を言語とする．この時， L_1 と L_2 の連接は L_1L_2 （又は $L_1 \cdot L_2$）と書かれ，以下の様に定義される．

$$L_1L_2 = \{x_1x_2 \mid x_1 \in L_1 \text{且つ} x_2 \in L_2\}.$$

例えば， $l_1 = \{00, 100\}$， $L_2 = \{1111, 001\}$ であるなら， $L_1L_2 = \{001111, 00001, 1001111, 100001\}$ である．要するに L_1 に入っている列と L_2 に入っている列の連接の全体である．

練習問題 1.4　任意の言語 L に対して $L\phi = \phi$ であることを示せ．

解答　まず， ϕ は空集合であることを思い出して欲しい．上の言語の連接の定義をよく見て頂きたい． L_1L_2 に含まれる列は x_1x_2 の形をしていて，その x_1x_2 が L_1L_2 に入るためには， x_1 が L_1 に入っていて且つ x_2 が L_2 に入っていなければならない．しかし，今 L_2 が空集合なので，そのような x_2 は存在しない．従って，「且つ」の条件が成立しないので，そのような x_1x_2 で L_1L_2 に入るものは存在しないということになる．従って，結果は勿論空集合である．このことと， $L\{\epsilon\} = L$ であることと混同しないで欲しい．更に老婆心で付け加えるなら， $L\{\epsilon\}$ を $L\epsilon$ と書かないで欲しい．言語と列の連接は定義されていないのである．（解答終わり）

列の連接の時と同じ様に $L^2 = LL$ や $L^3 = LLL$ の様な簡略表現を多用する．また，特別な場合として， L^1 と L^0 はそれぞれ L と $\{\epsilon\}$ を表すものと約束する．ここで，重要な定義を与える． L を言語としたとき，言語 L^* を以下で定義する．

$$L^* = L^0 \cup L^1 \cup L^2 \cup L^3 \cup \cdots.$$

つまり， L^* は L^i という形で与えられる言語全てを（つまり，任意の非負整数 i に対して）部分集合として含んでいる．この * をスター演算と呼ぶことがある．以前アルファベット（つまり，記号の有限集合）Σ に対して Σ^* を定義したが， Σ を記号の集合と見ずに，長さ 1 の列の集合と見ることによって，上の定義に矛盾していないことが分かる．このスター演算は結構強力で，これのみを使ってある種の無限言語を表現することができる．

練習問題 1.5　$L = \{a, bc, bcc, bccc\}^*$ はどんな言語かを分かりやすく説明せよ．

解答　L には定義より明らかに ϵ が入っているので， ϵ でない列 x が満たすべき条件を考えて見よう．答えを言ってしまえば， x が L に入るための必要十分条件は以下の 4 条件を満たすことである． (i) x の最初の記号は a 又は b である（つまり c ではない）． (ii) x に a が現れたら次の記号は（もしあれば） a 又は b である． (iii) x に b が現れたら次の記号は必ず c である． (iv) c は 3 個以上続かない（x は $cccc$ を部分列として含まない）． x が L に入るなら，以上の 4 条件を満たすという方向の証明は簡単であろう．定義より， x が L に入るなら，ある整数 i に対して， $x \in \{a, bc, bcc, bccc\}^i$ である．例えば $i = 3$ なら，

$$x \in \{a, bc, bcc, bccc\}\{a, bc, bcc, bccc\}\{a, bc, bcc, bccc\}$$

であるから， $x = x_1x_2x_3$ の形に書くことができて， x_1， x_2， x_3 のそれぞれは $\{a, bc, bcc, bccc\}$ の要素である．上の 4 条件がいずれも成立することはほぼ明らかであろう．

より困難なのは逆の証明である．これは上の 4 条件の内で一つでも満たされなければもはや x は L に入らないことを言えばよい．そのために，例えば，(ii) を満たさない，つまり，$x = y_1acy_2$ と書けるのに x が L に入ると仮定して見よう．$x \in L$ であるから，上で述べた様に $x = x_1x_2\cdots x_ix_{i+1}\cdots$（但し，各 x_i は $\{a, bc, bcc, bccc\}$ の要素）と書ける．y_1acy_2 の a が，ある x_i の最後の要素である場合とそうでない場合を考えると，いずれの場合も矛盾が導ける．前者の場合は，次の x_{i+1} が c で始まらねばならないが，$x_{i+1} \in \{a, bc, bcc, bccc\}$ に矛盾する．後者の場合は，ある列 α, β に対して，$x_i = \alpha ac\beta$ と書けることになり，これも $x_i \in \{a, bc, bcc, bccc\}$ であることからあり得ない．他の場合の解析は，読者にお任せする．なお，上で「(ii) を満たさないなら x は L に入らない」ことを直接証明するのではなく，「(ii) を満たさず且つ x が L に入る」と仮定して矛盾を導いた．このような証明法を背理法と呼んでしばしば利用する．（解答終わり）

練習問題 1.6　長さが奇数の $\{0, 1\}$ 上の列全体からなる言語を適当な方法で表現せよ．

解答　長さが偶数の列全体ならより易しいかもしれない．つまり，$\{00, 01, 10, 11\}^*$ で良いであろう．そこで奇数にするためには 1 個の 0 か 1 の記号を追加すれば良い．つまり，$\{00, 01, 10, 11\}^*\{0, 1\}$ とすれば良い．$\{0, 1\}$ 上の列全体から偶数長の列を除くという考えでも良い．その場合は，$\{0, 1\}^* - \{00, 01, 10, 11\}^*$ とすれば良い．集合 A と B に対して，$A - B$ は，A に入っていて，B に入っていない要素の全体を表す（必ずしも $A \supseteq B$ で無くても良い．）（解答終わり）

最後に，言語とは直接関係ないが数学的帰納法とともにこの分野の代表的証明テクニックである「鳩ノ巣原理」を説明しておく．鳩ノ巣原理そのものはいたって簡単である．鳩が $n + 1$ 匹いて，巣箱が n 個しかないなら，全ての鳩を一匹づつ巣箱に入れることは不可能であるということを言っているにすぎない．問題は応用であって，例えば以下の様に応用する．

練習問題 1.7　n を *1* 以上の整数とする．A を $A \subseteq \{1, 2, \cdots, 2n\}$ である大きさ $n + 1$ の集合とする（有限集合の大きさとはその集合に含まれている要素の数のことである）．この時，条件を満たすどんな A に対しても，x_1 が x_2 を割りきる様な *2* つの数 x_1 と x_2 が A に含まれていることを示せ．

解答　$A = \{a_1, a_2, \cdots, a_{n+1}\}$ とする．各 a_i を 2 で割れるだけ割って，$a_i = 2^{b_i} \cdot p_i$（ただし，p_i は奇数）とおく．p_i は奇数であるから，$p_i \in \{1, 3, \cdots, 2n - 1\}$ である．この集合の要素数は n である．しかし，A の要素数は $n + 1$ であるから，少なくとも 2 つの a_i と a_j が存在して，対応する p_i と p_j の値が等しくなる．（ここで鳩ノ巣原理を使用していることに注意されたい．つまり，要素数が $n + 1$ で，それらの取りうる異なった値の個数が n であるなら，少なくとも 2 つの要素が同じ値を取らざるを得ないのである．）つまり，

$$a_i = 2^{b_i} \cdot p_i,\ a_j = 2^{b_j} \cdot p_j,\ p_i = p_j$$

とかける．a_i と a_j において，小さい方が大きい方を割りきることは上の式から明らかである．（解答終わり）

1.4　まとめと文献

前書きでお断りしたように，ここで大学における基礎科目の概略と特色，どんな教科書があるか，更には限られた時間配分の中で基礎科目をいかに勉強したら良いか等々について，筆者の個人的考えも含めて述べてみたい．

情報工学や計算機科学等の名称を冠した学科のカリキュラムで「基礎」に分類される科目としては，アルゴリズムとデータ構造，アルゴリズムの設計と解析，オートマトンと言語理論，アルゴリズムと計算量の理論，等々の名称が思い浮かんでくる．実は前二者と後二者ではちょっと性質が違っていて，前二者は良いアルゴリズムを設計（そしてその優秀性を検証）するために必要な勉強という位置づけである．後二者はより抽象的と言っても良いかもしれない．つまり，そう言ったアルゴリズムの良し悪しをできるだけ数学的に論じたい時に，どのような理論体系が必要になってくるかを勉強する場であるという捉え方ができる．

後者は更に 2 つの系統がある．一つは伝統的な言語・オートマトン理論で，他方はいわゆる計算量理論と呼ばれる，計算時間や記憶容量の議論を中心としたどちらかと言えばより新しい学問体系である．勿論，これらは完全に分離できるものではなく，言語・オートマトン理論から計算量理論へと発展していくのが自然な勉強の道筋である．これらの勉強のために 4 単位（つまり，90 分授業がおよそ 30 回）が用意されているなら理想的である．つまり，前半の 2 単位を言語・オートマトン理論に，後半の 2 単位を計算量理論に当てることができるからである．

この場合の教科書の選択は困難ではない．言語・オートマトン理論の教科書，アルゴリズムと計算量の理論の教科書，いずれも数多くが出版されている．例えば，

・ホップクロフト, ウルマン（野崎他訳），言語理論とオートマトン，サイエンス社, 1971.

は古い様に見えるかもしれないが，やはり前者の教科書としては定番の位置を失っていないと思う．立派な本である．若干量が多いので， 15 週の講義に納めるためには適当に取捨選択する必要があるかもしれない．勿論，他にも数多く存在して，書店で「オートマトン」というキーワードで見繕って頂ければよいであろう．そのタイトルから想像される様に，本書もまたこのカテゴリの教科書である．

後半の 15 週のための教科書も選択に困るくらい存在する．世界的大家の本としては，

・ H. Lewis, C. Papadimitriou, Elements of the theory of computation, Prentice-Hall, 1981.

・ C. Papadimitriou, Computational complexity, Addison-Wesley, 1994.

・ Michael Sipser, Introduction to the theory of computation, PWS Pub., 1997.

あたりがすぐに思いつく．拙著で恐縮であるが

・岩間，アルゴリズム理論入門，昭晃堂， 2001.

も 15 週を標準とした講義の教科書として利用してもらうことを目的にしている．

しかし， 30 週でなく 15 週しか用意されていない（つまり 2 単位）場合もかなり多いであろう（各大学，学科によってポリシーが違っているから講義時間の割り振りは学部基礎科目においても当然違ってくる）．その場合にまんべんなく 30 週を 15 週に縮めるのはかなり困難であるし，賢明とはいえない．そこで，「オートマトン」か「計算量」かのいずれかを選択するわけである．しかし，前述の様にこれらはオーバラップしている部分もあるので，どちらかをメインにしているという言い方の方が適当である．回りくどい説明になってしまったが，本書は「オートマトン」をメインにして「計算量」については極めて基本的な事柄を挿入するにとどめた．逆に上の「アルゴリズム理論入門」では「オートマトン」は「計算量」を議論するために必要とな

る最低限にとどめた．手前味噌で赤面のいたりであるが，30 週の場合であれば，本書を前半，「アルゴリズム理論入門」を後半に利用して頂ければ望外の幸せである．15 週の場合はいずれかを選択して頂けることを願っている．

最後にいくつかのギリシャ文字をまとめておく．数学的記号は非常に重要で，ある一定の約束がある（たとえば，アルファベットを意味する記号としては Σ が極めて頻繁に利用される）．記号だからきっちり断わっておけば何を使っても良いとは絶対に考えないで欲しい．その意味で，本書でもできるだけ標準的且つある意味で「美しい」記号を使うようにつとめた．そのためにはギリシャ文字は欠くことができないのである．

α（アルファ），β（ベータ），Γ, γ（ガンマ），Δ, δ（デルタ），ϵ（イプシロン）Θ, θ（シータ），Λ, λ（ラムダ），μ（ミュー），Σ, σ（シグマ），τ（タウ），Φ, ϕ（ファイ），Ψ, ψ（プサイ），Ω, ω(オメガ）．

理解度の確認

問　1.2 節でカッコのバランスした言語を論じた．そこではカッコの種類は (と) の 1 種類だけであったが，{ や }，[や] の様にいくつかの種類を使用することもできる．以上の 3 種類のカッコを使用した場合の，カッコのバランスの正しさを判定する方法を与えよ．なお，以上の 3 種類のカッコは特にレベルの違い（例えば，{ と } は (と) の必ず外側でなければならないと言った制限）は考えなくて良い．単に対応するカッコの種類が同じなら良い．

問　以下はどんな言語であるか．推測して分かりやすく説明せよ．

$$
\begin{aligned}
L_1 &= \{1, 010, 00100, 0001000, \cdots\} \\
L_2 &= \{\epsilon, 00, 11, 0000, 0101, 1010, 1111, 000000, 001001, \cdots\} \\
L_3 &= \{1, 010, 011, 110, 111, 00100, 00101, 00110, 00111, 01100, 01101, 01110, \\
&\quad 01111, 10100, 10101, 10110, 10111, 11100, 11101, 11110, 11111, 0001000, \cdots\}
\end{aligned}
$$

問　$\{0, 1\}$ 上の列で部分列として uu（$u \neq \epsilon$）の形を含まないものは有限個しか存在しないことを証明せよ（例えば列 011 は既に uu の形の部分列を含んでしまっている）．uuu を部分列として含まない場合はどうか．

問　列 x で，$x = x^{\mathrm{R}}$ を満たすものを回文と呼ぶ．回文の再帰的定義を与えよ．

問　以下で，全ての馬の色が等しいことを数学的帰納法で証明する．勿論何かが間違っているのであるが，どこが間違っているか．(i) 馬が 1 匹の時は勿論全ての馬の色は等しい．(ii)n 匹の時に全ての色が等しいと仮定しよう．$n+1$ 匹の馬を考える．(iii) どれでも良いから 1 匹の馬（h_1 とする）を排除すると，n 匹になるので，帰納法の仮定よりその n 匹の馬の色は全て等しい．(iv) さて，今排除した馬を集団に戻し，今度は別の馬（h_2 とする）を排除して再び帰納法の仮定を使うと残った n 匹の馬の色は等しい．いずれの場合も排除されなかった馬を考えると，この馬の色は (iii) より h_2 の色に等しく，(iv) より h_1 の色にも等しい．従って，h_1 の色と h_2 の色は等しく，更に他の全ての馬の色に等しい．

問　練習問題 1.7 では A として大きさ $n+1$ の集合を考えたが，大きさを n とするともはやこの

性質は成り立たないことを示せ.

問　あなたが 77 日間連続でテストを受けないといけないとしよう．但し，1 日最低 1 個のテストを受けなければならず，全体では 132 個より多くのテストを受けてはいけない．この 2 条件を満たすようにテストを受けたとすると，どんなテストの受け方をしても，ある連続した何日かに丁度 21 個のテストを受けることになってしまうことを証明せよ．（ヒント：i 日目に a_i 個のテストを受けたとしよう（$1 \leq i \leq 77$）．つまり，$a_i \geq 1$，$a_1 + a_2 + \cdots + a_{77} \leq 132$ を満たす様に，a_1 から a_{77} を決めたとする．この時，$c_i = a_1 + \cdots + a_i$，$d_1 = c_i + 21$, $1 \leq i \leq 77$ と置く．$c_1, \cdots, c_{77}, d_1, \cdots, d_{77}$ は全部で 154 個の数であり，それらの取りうる範囲は 1 から $132 + 21 = 153$ までである．鳩ノ巣原理を使え．）

問　k 記号のアルファベット $\Sigma_k = \{a_1, a_2, \cdots, a_k\}$ 上の列 x で以下の性質を持つものを考える．すべての i と j（ただし $i > j$）の組みに対して x は $a_i a_j$ か又は $a_j a_i$ を部分列として含む．$k = 7$ の時にそのような性質を満たす x で長さが最小のものを求めよ．一般の k に対してはどうか．特に最小の x の長さがどのようにして表せるかを考えよ．

第 2 章

正規表現と有限オートマトン

前章で述べた様に本書の主題は無限集合である言語をいかに表現するかである．有限個の列しか含まない有限言語であれば「書き下す」という方法が使えるのであるが，無限言語の場合はそれは勿論不可能である．そこで，例えば，前章の章末問題の様に列の短い方からの何個かを書いて推測させるといった方法を使用するのであるが，これは勿論数学的厳密さに欠ける．そこで一般にはその言語に属す列が満たすべきルールを上手に書いて，それをもって言語の表現とするという手法が使われる．文法がその代表で，例えば，受験勉強で英文法の分厚い本を勉強された（あまり良くない）思い出があるであろう．

我々が対象としている形式言語においても文法は強力な手段である．しかし，文法の導入は次章以降に延して，この章ではより簡潔な（しかし，数学的厳密さを備えた）正規表現と呼ばれるシステムを紹介する．次に有限オートマトンと呼ばれる別の表現法を紹介する．これら 2 つの方法は見掛け上はかなり異なっているのであるが，実は等価である．その等価性の証明がこの章の最も重要な目的である．

2.1 正規表現

前章でスター演算を導入したが，これのみでも，練習問題 1.5 に見られるように結構面白い言語を表現できる．正規表現では実はこのスター演算が重要な役割を演じる．なお，正規表現は UNIX でも利用されているが，ある性質をもった列を上手に表現したいという精神は同じであるが，この章でのべる正規表現は UNIX のそれとは若干異なっている（より数学的とでも言おうか）ことを注意しておく．

Σ をアルファベットとしたとき，Σ 上の**正規表現** S と S が表現する言語 $L(S)$ は以下の様に（再帰的に）定義される．（慣れていない読者の方は戸惑うかもしれないが，ちょっとの間我慢して欲しい．後ですぐに例を使って説明する．）

(1) ϕ は正規表現であり，$L(\phi) = \phi$ である．

(2) ϵ は正規表現であり，$L(\epsilon) = \{\epsilon\}$ である．

(3) $a \in \Sigma$ なら，a は正規表現であり，$L(a) = \{a\}$ である．

(4) R_1 と R_2 が正規表現なら，(R_1+R_2)，$(R_1\cdot R_2)$（これは (R_1R_2) の様に簡単に書いてもよい），(R_1^*) のそれぞれもまた正規表現であり，$L((R_1+R_2))=L(R_1)\cup L(R_2)$，$L((R_1\cdot R_2))=L(R_1)L(R_2)$，$L((R_1^*))=(L(R_1))^*$ である．

このような書き方に慣れておられない方は何を言っているか分からないと仰有るかもしれない．例えば，(1) は，ϕ という 1 記号そのものが正規表現であり，それは空集合という言語を表す，とよむのである．つまり，最初の ϕ は正規表現に利用する特別な記号，2 番目の ϕ は従来からの空集合を表すといういみで，正確に言えば別物である．(4) では，2 つの正規表現があったときそれらを $+$ で結んだもの（つまり，正規表現もまた我々の言葉で言えば「列」なのである）をカッコでくくったものもまた正規表現であることを言っている．例えば，$((0(1^*))+0)$ は $\{0,1\}$ 上の正規表現であることが以下の様に分かる．まず，1 は (3) によって正規表現である．次に,(1^*) は (4) の 3 番目のルールによって，正規表現である．よって，$(0(1^*))$ は (4) の 2 番目のルールで正規表現である．最後に，全体が (4) の最初のルールで正規表現になることが分かる．

カッコが多すぎると見ずらいので，演算に優先度を設けることによって，省略できるようにしよう．優先度は *，$\cdot$，$+$ の順で高いことにする．そうすれば，例えば上の正規表現 $((0(1^*))+0)$ は単に $01*+0$ と書いても混乱は生じないことになる．更に，演算 $\cdot$ と $+$ は明らかに結合律が成り立つので，例えば，$((0+1)+2)^*$ の代わりに $(0+1+2)^*$ と書いても良い．更には $R^+=RR^*$ や $R^5=RRRRR$ の様な表現も時々用いる．

練習問題 2.1　次の正規表現はどんな言語を表現しているか．*(i)* $R_1=\phi^*$．*(ii)* $R_2=((0+\epsilon)^*+(001+11)^*)\phi$，*(iii)* $R_3=(0+1)^*000(0+1)^*$．

解答　(i) 正規表現の (4) の 3 番目のルールを思い出して欲しい．このルールを適用すると $L(R_1)=\phi^*$ となる．ここで，ϕ は勿論空集合（列を 0 個含む言語）であり，言語（集合）L に対する * 演算は前章で定義した様に，$L^*=L^0\cup L^1\cup L^2\cup L^3\cup\cdots$ である．ここで，L^0 は定義より L がどんな集合であっても $\{\epsilon\}$ であるから，$L(R_1)$ はとりあえず ϵ を含むことが分かる．その後の L^1 や L_2 は L 自身が空集合なので空集合である．結局，$L(R_1)=\{\epsilon\}$ である．(ii) R_2 を良く見ると，正規表現 $R=((0+\epsilon)^*+(001+11)^*)$ に対して，$R\phi$ という正規表現になっている．(4) の 2 番目のルールから，これは $L(R)$ と空集合を連接して言語になる．言語の連接の定義から，任意の言語に空集合を連接すると結果が空集合になることを以前に言った．つまり，$L(R_2)=\phi$ である．(iii) $L((0+1)^*)=\{0,1\}^*$ である．つまり，$L(G_3)$ はどこかに 3 個以上の 0 が連続して現れる $\{0,1\}$ 上の列の全体を表す．（解答終わり）

練習問題 2.2　$R_5=c^*(a+bc^*)^*$ はどんな言語を表現するか．

解答　答えを先に言ってしまえば，$L(R_5)$ は部分列 ac を含まない $\{a,b,c\}$ 上の列の全体を表す．以下で理由を説明する（証明というほど正式のものではないが）．正規表現の定義から

$$\begin{aligned}L(R_5) &= L(c^*(a+bc^*)^*)\\ &= L(c^*)(L(a+bc^*))^*\\ &= c^*\{a,b,bc,bcc,bccc,\cdots\}^*\end{aligned}$$

である．従って，$A=\{a,b,bc,bcc,bccc,\cdots\}$ と置くと，$L(R_5)$ に入る列 x は最初に何個か（0 個以上）の c の後に，A に属する列を何個か（0 個以上）連接したものである．つまり，

$$x=c^iy_1y_2\cdots y_j\cdots y_n,\ \ y_j\in A$$

と書ける．ここで，最初の c^i からは明らかに部分列 ac は現れない．つぎに， c^i と y_1 の境界でも現れないのは明らかである．次に y_1 の内部でも明らかに現れない．最後に y_1 と y_2 の境界であるが， y_2 は必ず a 又は c で始まるので，やはり部分列 ac は現れようがない．後は同様なので，こうして $x \in L(R_5)$ なら， x には部分列 ac は現れないということが言えた．

次に逆である．つまり， $\{a, b, c\}$ 上の列 x に部分列 ac が現れないなら $x \in L(R_5)$ であることを言わなければならない．これは次の様に考えよう．列 x を左から見ていって，記号 c の前以外は全て「切目」を入れる．例えば，

$$x = ccc|a|a|bccc|b|bc|a|bccc|b|b|$$

の様に切目が入る． c の前だけ切目が入っていないことが分かるであろう．さて，ある切目と切目の間の部分列を y としよう．つまり， $x = \cdots|y|\cdots$ と書けたとする．以下の場合が考えられる．

(場合 1） y の右端の記号が a の場合．その場合，その a の左にも切目が入るので結局 $y = a$ である．

(場合 2） y の右端の記号が b の場合．同様に $y = b$.

(場合 3） y の右端の記号が c の場合．この c から左に見ていくと， c の左には切目が入らないし，次がまた c の場合も同様である．もし， b が来るとその左で切目が入る． x の条件から c の左には a は来ない．従って，この場合， $y = bc^i$， $i \geq 1$ と書ける．

いずれの場合も， $y \in A$ であることが分かる．従って， $x \in \{c\}^* A^*$ と結論でき，これが証明したかった事である． （解答終わり）

練習問題 2.3 $\{0, 1\}$ 上の列 x の 1 の個数を $\sharp_1(x)$ と書くことにしよう（$\sharp_0(x)$ も同様に 0 の個数である）． $L(R) = \{x \,|\, x \in \{0,1\}^*$ 且つ $\sharp_1(x)$ が偶数$\}$ であるような正規表現 R を求めよ．条件を $\sharp_0(x)$ と $\sharp_1(x)$ が共に偶数になることと変更するとどうか．

解答 例えば，列

$$x = 001001011101000.$$

は偶数個の 1 を含んでいる．この列はまた，

$$x = 0^2(10^2 10^1)(10^0 10^0)(10^1 10^3).$$

と書けることも分かって欲しい．そうすれば，もう答えにたどり着くのは容易であろう．すなわち， $R = 0^*(10^*10^*)^*$ である．次に， 0 と 1 の個数が共に偶数の場合であるが，これは簡単な応用である．全体の長さは当然偶数になるので， 2 個づつ記号を区切っていくと，その 2 個の記号は $00, 01, 10, 11$ のいずれかになる． 0 と 1 の個数が共に偶数になるための必要十分条件は 01 と 10 の個数の和が偶数になることであることは容易に分かろう．従って，上の 1 の個数が偶数の場合の 1 を $(01 + 10)$ で置き換えるという考えがでてくる．詳細は読者が自ら考えて欲しい．（解答終わり）

正規表現に関する重要な性質は何と言っても後述する有限オートマトンとの等価性なのであるが，そのほかに関しては，以下の基本的なものだけ述べておく．

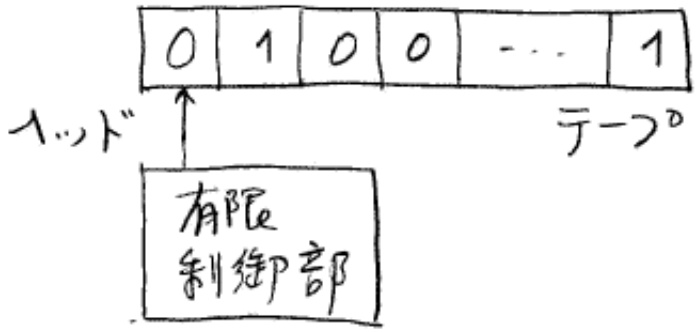

図 2.1 有限オートマトン

定理 2.1　言語 L_1 と L_2 が正規表現で表現できるなら，$L_1 \cup L_2$，L_1L_2，L_1^* のそれぞれのまた正規表現で表現できる．

証明　正規表現の定義の (4) から明らかである．（証明終り）

2.2　有限オートマトン

前節の正規表現がどちらかというと，言語に属する列を「生成する」というイメージであったのに対し，ここでの有限オートマトンは，与えられた列が言語に属するかどうかを「判定する機械」のイメージである．従って，物理的動作と結び付けると分かりやすい．将来もっと複雑なオートマトンモデルを導入して計算機のモデルであると主張する場面も出てくると思う．しかし，この「物理的側面」は便宜上のものであることを肝に銘じてほしい．オートマトンはあくまで，言語を表現するための数学的手段なのである．論理回路等の講義で順序機械を勉強している読者の方は特に注意して欲しい．物理的に動作するものと決めつけてしまうと，将来，特に非決定性オートマトンを導入する時に混乱してしまう．繰り返すが，オートマトンはあくまで数学的モデルである．

図 2.1 に有限オートマトンの概念図を示す．テープと有限制御部からなり，ヘッドがテープを左から右へスキャンする．テープはマス目にわかれていて，各マス目には記号が書いてある．従って，テープ全体で 1 個の列になる．有限オートマトンはテープをスキャンした後で，そのテープ（列）が言語に属しているかどうかを判定することによって言語を表現する．前節の正規表現はどちらかというと言語に属する列を生成するというイメージなので，表現の仕組みはちょっと違っていると言える．

有限オートマトン（fa）は $M = (K, \Sigma, \delta, s_0, F)$ で与えられる．ここで，K, Σ，$F \subseteq K$ はそれぞれ有限集合である．K の要素は**状態**，Σ の要素は**入力記号**，F の要素は**受理状態**と呼ばれる．s_0 は K のある要素で**初期状態**と呼ばれる．δ は $K \times \Sigma$ から K の中への写像で，**状態遷移関数**と呼ばれる．

図 2.2 に fa の例を与える．M_1 で表す．定義によれば，K, Σ，F，s_0，δ を具体的に与えれ

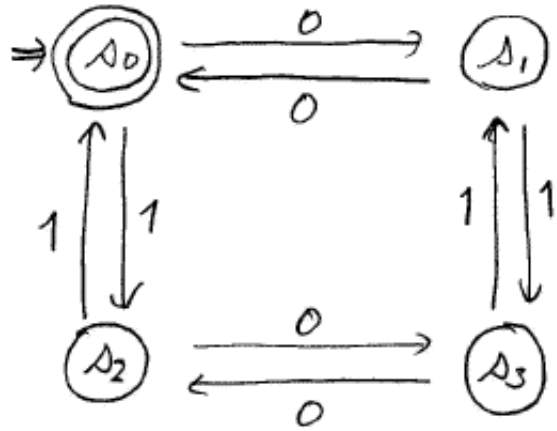

図 2.2 有限オートマトン M_1 の状態遷移図

ば良いのであるが，無限のものは何もないので，全て「書き下す」ことが可能である．図 2.2 はそれら全てをあるルールによって書き下したもので，状態遷移図と呼ばれる．状態は s_0 から s_3 の 4 個，入力記号は 0 と 1，初期状態は二重矢印で表されている s_0，受理状態は二重丸で表されている s_0（この例の場合受理状態がただ 1 個でそれが初期状態にもなっている），状態遷移関数は状態から状態への矢の形で表されている．例えば，$\delta(s_0, 0) = s_1, \delta(s_0, 1) = s_2$ である．これは，状態 s_0 で記号 0 を読めば s_1 に状態を変化することを示している．

状態 s_0 で 3 個の記号 110 をこの順で読めば（これを記号列 110 を読むという），状態は，s_2，s_0 を経て s_1 に至る．このことを表すために，$\hat{\delta}$ という写像を導入する．つまり，$\hat{\delta}(s_0, 110) = s_1$ となる様に定義するのである．定義は以前の様に再帰的定義を利用する．すなわち，

$$\text{任意の} s \in K, a \in \Sigma, x \in \Sigma^* \text{に対して，} \hat{\delta}(s, \epsilon) = s,\ \hat{\delta}(s, xa) = \delta(\hat{\delta}(s, x), a)$$

とすれば良い．$\hat{\delta}(s, x)$ は x を 1 記号とすれば，δ と同じになる．従って，δ と上の山の印（これをハットと呼ぶ）を省いても混乱が生じることはないので以下ではそのようにする．（従って，δ の定義域が $K \times \Sigma$ から $K \times \Sigma^*$ へ「自然に」拡張されたことになる．）与えられたテープの記号列が x であった時，$\delta(s_0, x) = s_0$ であるなら，つまりテープを読み終った時の状態が受理状態であるなら，その列 x は**受理**される．M_1 が受理する列の全体を M_1 が**認識**する言語という．

練習問題 2.4　M_1 はどんな言語を認識するか．

解答　実は前節の練習問題 2.3 でやった，0 と 1 の個数が共に偶数となる $\{0, 1\}$ 上の列の全体が答えである．これを数学的帰納法で証明しよう．まず，証明すべき命題であるが，それは以下のようなものである．「列 $x \in \{0, 1\}^*$ に対し，$\delta(s_0, x) = s$ とすると，$s = s_0$ であるなら x の 0 と 1 の個数は共に偶数，$s = s_1$ なら 0 の個数のみ奇数，$s = s_2$ なら 1 の個数のみ奇数，$s = s_3$ なら共に奇数である．」列 x の長さが 0 のときは，定義から x を読んだ後の M_1 の状態は s_0 である．また長さ 0 の列の 0 と 1 の個数は共に偶数であるから，命題は成立している．長さ n（≥ 0）の時命題が成立すると仮定して，長さ $n+1$ の列 y をを考えよう．y の長さは 1 以上であるから，ある $x \in \{0, 1\}^*$, $a \in \{0, 1\}$ に対して $y = xa$ と書ける．ここからは多くの場合

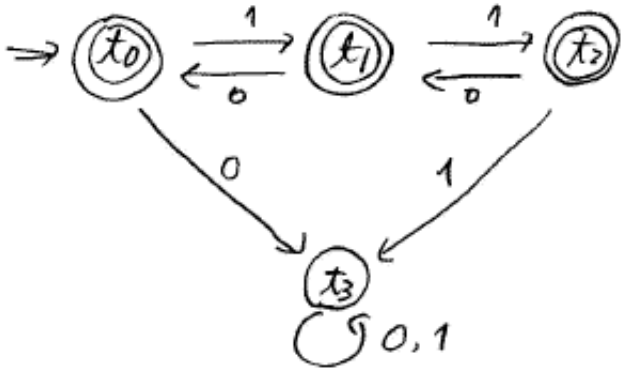

図 2.3 L_1 を認識する fa

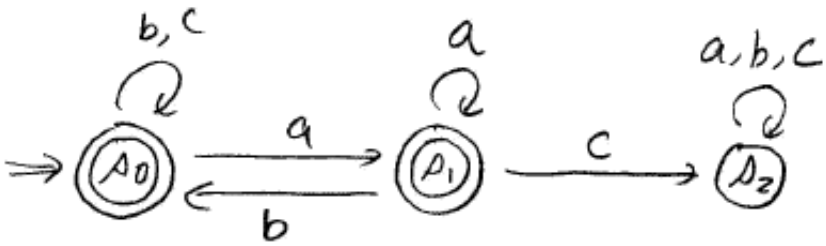

図 2.4 L_2 を認識する fa

を考える必要があるが，例えば，x では 1 の個数のみが奇数で，$a = 0$ としよう．すると，y では 1 も 0 も共に奇数になる．x の長さは n であるから帰納法の仮定が使えて，1 の数が奇数なら x を読み終わった後の M_1 の状態は $s = 2$ である．状態遷移図から，s_2 で $a = 0$ を読むと s_3 に遷移する．こうして，y を読んだ後の状態が s_3 であることが分かった．0 も 1 も奇数個の y を読み終わったあとの状態が s_2 であることが分かったがこれは命題に矛盾していない．他の場合を全て調べれば列 y においても命題が成立していることが分かる．（解答終わり）

練習問題 2.5　言語 $L_1 = \{x \mid x \in \{0,1\}^*$ 且つ任意の x のプレフィックス y に対して，$0 \leq \sharp_1(y) - \sharp_0(y) \leq 2\}$ と $L_2 = \{x \mid x \in \{a,b,c\}^*$ 且つ x は部分列 ac を含まない$\}$ を認識する fa を与えよ．但し，プレフィックスとは列の先頭から始まる部分列のことである（つまり，$y = x_1x_2$ と書けるとき，x_1 は y のプレフィックスである）．

解答　図 2.3 と 2.4 にそれぞれ L_1 と L_2 を認識する fa の状態遷移図を示す．正しさはほぼ明らかであろう．なお，L_2 は練習問題 2.2 と同じ言語である．図 2.4 の fa を良く観察してもらうと練習問題 2.2 で与えた正規表現と異なった正規表現で同じ言語を表現するものが思い浮かばないであろうか．（解答終わり）

有限オートマトンに関する性質も最低限述べるにとどめる．ただ一つ，fa に関する便利な道具として，**直積オートマトン**を紹介しておこう．入力アルファベットが等しい 2 つの fa $M_1 = (K_1, \Sigma, \delta_1, s_{01}, F_1)$ と $M_2 = (K_2, \Sigma, \delta_2, s_{02}, F_2)$ の直積とは，2 つの fa を並列に動作させたシステムを模擬した有限オートマトンと考えれば良い．状態は 2 つの fa の状態の対 $(s_1, s_2) \in K_1 \times$

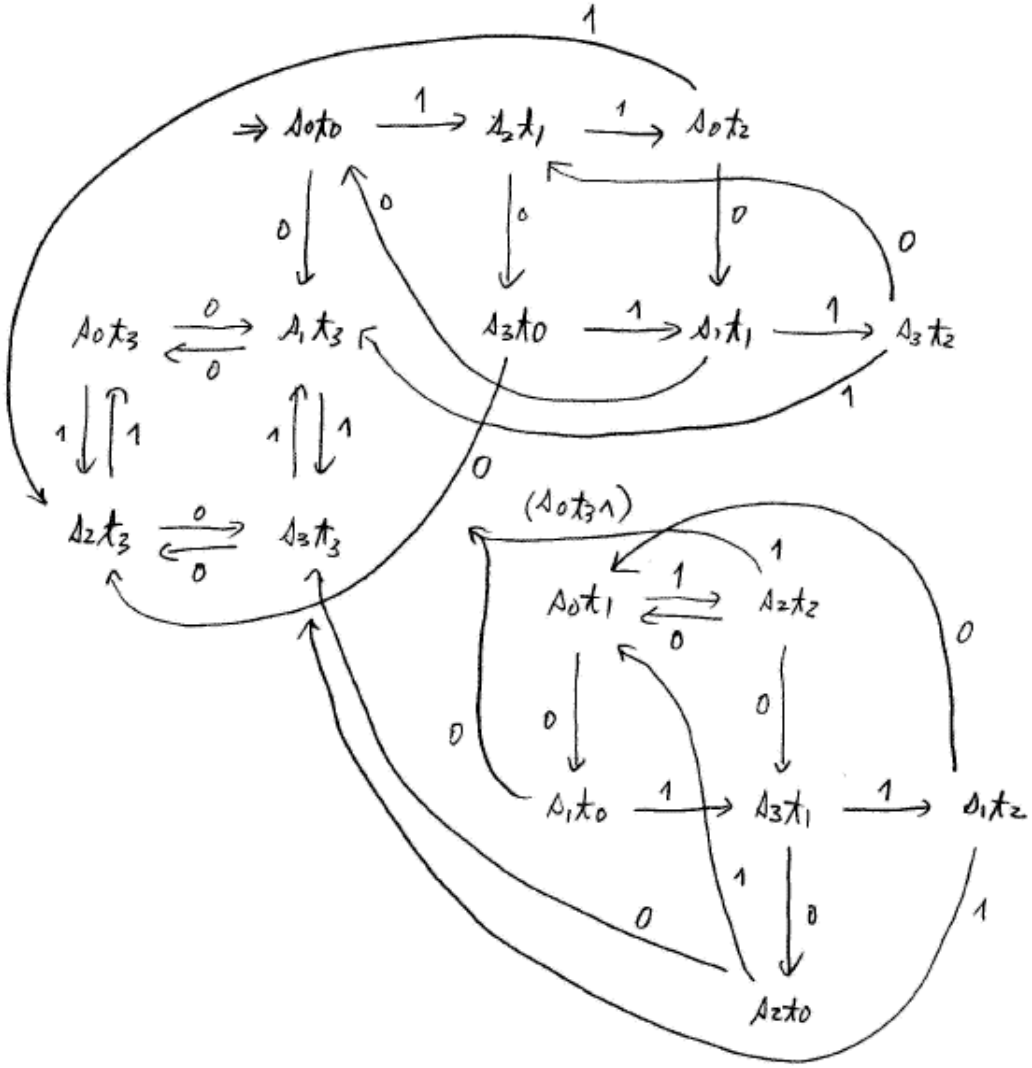

図 2.3-1 直積オートマトン

K_2 で，この状態で入力 $a \in \Sigma$ を読むと $(\delta_1(s_1, a), \delta_2(s_2, a))$ に遷移する．例えば，図 2.2 と図 2.3 の fa の直積は状態として $\{(s_1, t_0), (s_0, t_1), \cdots, (s_3, t_3)\}$ の 16 状態を有し，例えば (s_0, t_0) で 1 を読むと (s_2, t_1) に遷移し，更にそこで 1 を読むと (s_0, t_2) に遷移する．直積オートマトンの初期状態と受理状態は利用目的によって異なってくる．例えば，

定理 2.2　言語 L_1 と L_2 が *fa* で認識できるなら，$L_1 \cap L_2$ もまた *fa* で認識できる．

証明　L_1 と L_2 を認識する fa をそれぞれ M_1，M_2 とすると，それらの直積を作る．初期状態としては，M_1 の初期状態と M_2 の初期状態の対を選ぶ．受理状態としては，対 (s_1, s_2) の中で，s_1 が M_1 の受理状態で且つ s_2 が M_2 の受理状態である様なものを全て選ぶ．こうして作った直積オートマトンが $L_1 \cap L_2$ を認識することは明らかである．つまり，$L_1 \cap L_2$ は有限オートマトンで認識できる．なお，図 2.2 と図 2.3 の fa に対してこのようにして構成した直積オートマトンは図 2.3.1 の様になるので参考にして欲しい．　（証明終り）

fa は有限個の状態を有するが，その状態の個数が記憶容量と考えられ，同じ言語を認識する

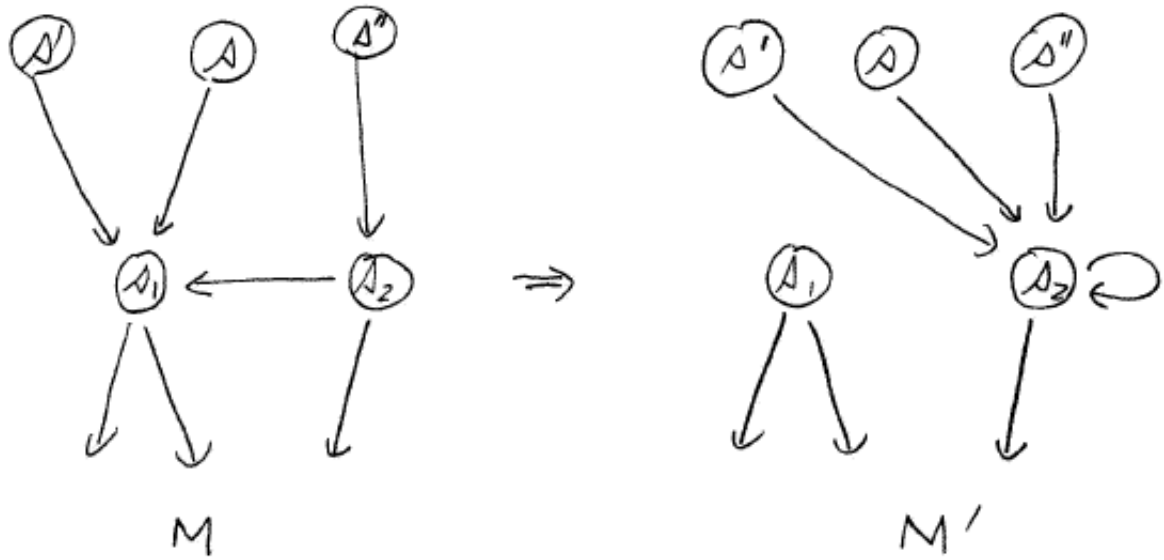

図 2.3-2 オートマトンの状態最少化

限り少なければ少ないほど効率が良いと言える．従って，冗長な状態を排除することが望まれるが，必要の無い状態としてすぐに気づくのが初期状態から到達できない状態である．図 2.3.1 の直積オートマトンを見て欲しい．この fa の初期状態は (s_0, t_0) であるが，そこから例えば (s_0, t_1) はどんな列を読んでも到達できないことが容易に観察できるであろう（状態遷移図で (s_0, t_0) から矢印をどのようにたどろうとも (s_0, t_1) には到達できない）．このような状態を削除しても認識する言語に変化が生じないのは明らかである．図 2.3.1 はこうして削除できる状態を 6 個含んでいることを読み取って欲しい．

そこで，以下では fa はこのよう初期状態から到達できない状態は含まないと仮定しよう．しかし，それでもなお冗長な状態が存在する可能性がある．このことを議論するために必要になるのが状態の等価性という概念である．有限オートマトン $M = (K, \Sigma, \delta, s_0, F)$ の 2 つの状態 s_1 と s_2 は，条件「任意の列 $x \in \Sigma^*$ にたいして，$\delta(s_1, x) \in F$ の時またそのときに限って $\delta(s_2, x) \in F$ である」を満たすとき**等価**であるという．

定理 2.3 *(i) fa M が等価な 2 個の状態を有するなら，M と同じ言語を認識し，状態数が M より 1 個少ない fa M' が存在する．(ii) fa M（上で仮定した様に初期状態から到達できない状態は存在しない）に対して，同じ言語を認識して状態数がより少ない fa M' が存在するなら，M には必ず等価な 2 個の状態が存在する．*

証明 (i) 等価な 2 つの状態を s_1 と s_2 とする．状態 s から s_1 への遷移が存在するなら，その様な遷移を全て s から s_2 への遷移へと変更してしまう（図 2.3.2 参照）．こうして，状態遷移図では s_1 への遷移が無くなってしまったので，その状態そのものを削除して状態数が 1 少ない fa M' を構成する．M' が M と同じ言語を認識することの証明は読者にお任せする．(ii) M と M' が同じ言語を認識し，M の状態数が M' の状態数より 1 多く，且つ M の任意の 2 つの状態は等価でないと仮定して見よう．更に，定理では M の状態は全て初期状態から到達できると仮定している（到達できない状態は初めから除いてしまえば良いので一般性を失わない）．

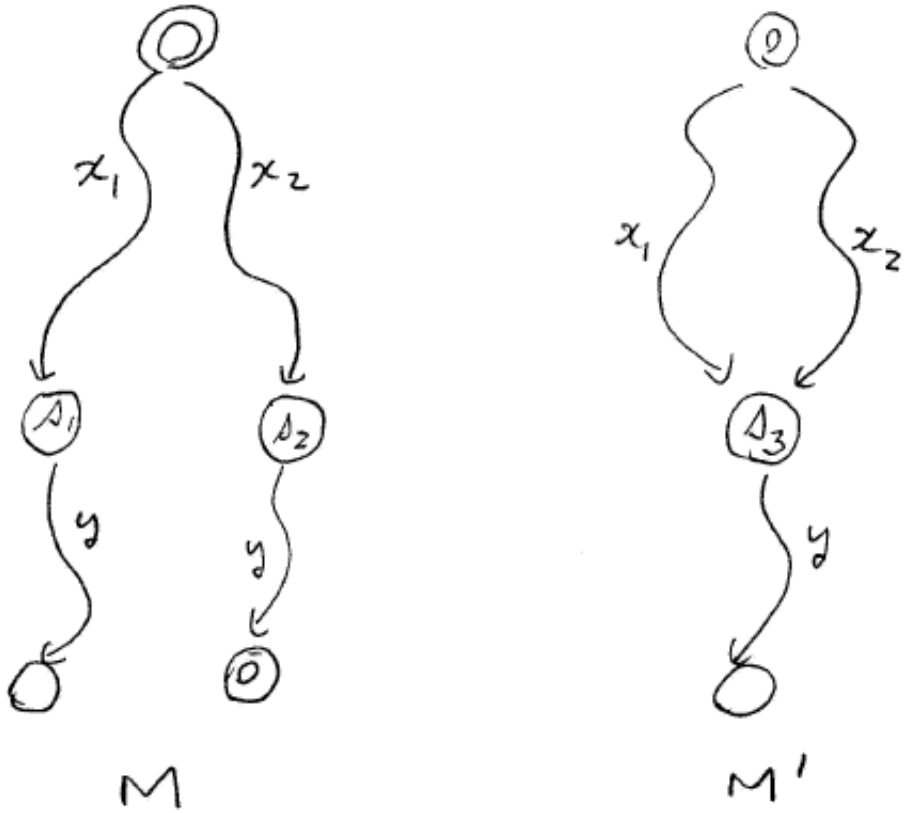

図 2.3-3 定理 2.3 の証明

M の状態数が M' の状態数より 1 多いので，1 章で述べた鳩ノ巣原理を使うと，「異なった 2 つの列 x_1 と x_2 で，初期状態から x_1 と x_2 を読ませると M では異なった状態 s_1 と s_2 に遷移するのに，M' では同じ状態 s_3 に遷移するような x_1 と x_2 が存在する（図 2.3.3 参照）」ことが言える（理由：M の状態数を n とすると，初期状態からそれらのことなった状態に遷移させる n 個の列が存在する．これらの列を M' に与えれば，M' は $n-1$ 状態しかないので，少なくとも 2 つの列によって同一の状態に遷移する）．仮定より，s_1 と s_2 は等価では無いので，ある列 y が存在して，s_1 から y を読むと受理状態に遷移し，s_2 から y を読むと非受理状態に遷移する．つまり，M では x_1y と x_2y は一方が受理されて，他方が受理されない．しかし，M' では明らかに双方の列が共に受理されるか受理されないかのいずれかである．これは M と M' が同じ言語を認識するという前提に反する．（証明終り）

この定理によれば，fa が与えれたら，その 2 つの状態の全ての組合わせ（有限通りしか存在しないのでシラミ潰しに実行できる）に対してそれらが等価かどうかを調べ，等価なら上の方法で一方を削減するという処理をそれ以上できなくなるまで続けると最小状態数の fa が得られることになる．問題は 2 つの状態が等価か否かをいかにして調べるかである．定義によれば，全ての列（無限個存在する）に対して調べなければならないので単純に定義通り調べることはできない．ここでもまた直積オートマトンが便利に利用できるのである．

練習問題 2.6　*fa M の 2 つの状態 s_1 と s_2 が等価か否かを調べる方法を与えよ．*

解答　M と M の直積オートマトンを作る（同一の 2 つの fa であっても直積オートマトンは問題なく構成できる）．初期状態は (s_1, s_2) として，まず，初期状態から到達できない状態は取

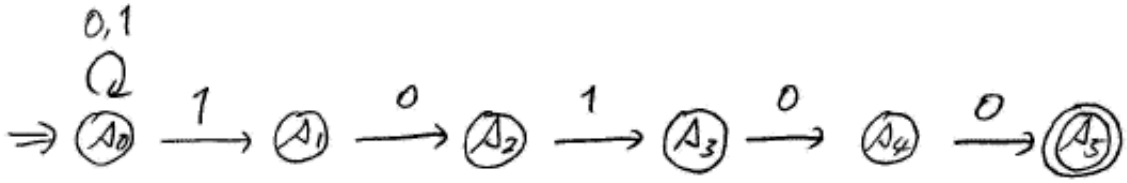

図 2.5 非決定性有限オートマトン M_2

り除こう．残った状態は初期状態から到達できるし，それらは (s_3, s_4) の様に対になっている．この状態を全て（有限個であることに注意）チェックして，対の一方が受理状態で他方が非受理状態であるような対が存在すればまたそのときに限って非等価と判定する．この方法の正しさの証明は読者にお任せする．（解答終り）

この節の最後に言語の補集合を考えよう．例えば，図 2.2 の M_1 は 1 の数と 0 の数が共に偶数の列の全体からなる言語（L_1 と置こう）を認識するが，この言語の補集合（$\{0,1\}^* - L_1$，つまり 0 の個数か 1 の個数が奇数である様な列の全体）はどんな fa によって認識できるであろうか．

定理 2.4　Σ 上の言語 L が *fa* で認識できるなら，L の補集合 $\Sigma^* - L$ もまた *fa* で認識できる．

証明　L を受理する fa を M とする．我々は，M で受理しない列を受理する fa をつくりたい．受理しない列というのはそれを読み終った時に非受理状態にある．つまり，M の受理状態と非受理状態を単に入れ替えれて M' とすれば，この M' が我々の目的を達成する，つまり $\Sigma^* - L$ を受理する fa になる．（証明終り）

2.3　非決定性有限オートマトン

非決定性有限オートマトンを導入する目的は 2 つある．一つは，この章の重要な目的である有限オートマトンと正規表現の等価性を証明するのに必要だからである．二つ目は「非決定性」の概念を最も簡単なオートマトンで早めに理解して欲しいからである．後でより複雑なオートマトンに対して非決定性のモデルを導入するが，有限オートマトンの場合の方が明らかに説明が楽である．

今までの fa（これを**決定性 fa，dfa** と呼ぶこともある）の場合は，現在の状態と読む入力記号が定まれば次の状態は一意に定まる．しかし，非決定性 fa の場合は次の状態として 2 つ以上の可能性があって良い．

非決定性有限オートマトン（非決定性 fa，nfa）は $M = (K, \Sigma, \delta, s_0, F)$ で与えられる．ここで，状態遷移関数 δ のみが決定性 fa の場合と異なり，$K \times \Sigma$ から K の部分集合族（K の全ての部分集合の集合）の中への写像である．

図 2.5 に nfa の例（M_2）を与える．状態 s_0 で入力記号 1 を読んだ時の遷移先が s_0 と s_1 の二

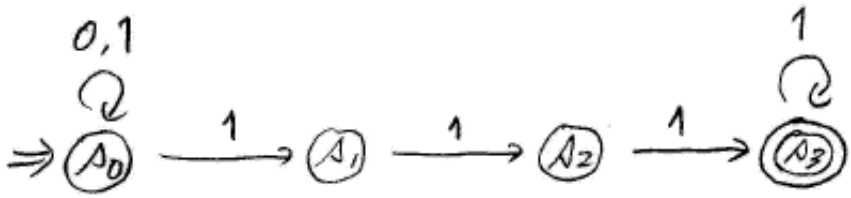

図 2.6 練習問題 2.6 の解答（非決定性 fa）

つ指定されている．つまり，$\delta(s_0, 1) = \{s_0, s_1\}$ である．また，s_1 で 1 を読んだときの遷移先は空集合になっている．つまり，$\delta(s_1, 1) = \phi$ である．

次に，nfa が認識する言語を定義する．そのために，以前と同様，δ の定義域を拡張して，$\hat{\delta} : K \times \Sigma^* \to 2^K$ を以下の様に定義する．$s \in K$ を状態，$x \in \Sigma^*$ を入力記号の列，$a \in \Sigma$ を（一つの）入力記号とした時，

$$\hat{\delta}(s, \epsilon) \equiv \{s\}$$

$$\hat{\delta}(s, xa) \equiv \{q \mid q \in \delta(p, a)\text{かつ}p \in \hat{\delta}(s, x)\text{であるような}p \in K\text{が存在する}\}$$

で定義される．例えば，M_2 の場合，$\hat{\delta}(s_0, 101) = \{s_0, s_1, s_3\}$ であることを確認してほしい．（先ず最初は $x = 10, a = 1$ とおく．こうして $\hat{\delta}$ の 101 に対する値を 1 つ短い列 10 に対する値を使って示すことができる．これを繰り返すのである）．

このように形式的定義はあくまで厳密なものであるが，やはり，その動作をより直観的に説明できると便利なことが多い．即ち，$\delta(s_0, 1) = \{s_0, s_1\}$ を「s_0 で入力 1 を読むと，状態 s_0 にも s_1 にもいずれにも遷移できる」と解釈すると都合が良い．これらの状態で更に 0 を読むと，s_0 からは s_0 へ，また s_1 からは s_2 へと遷移できる．つまり，s_0 で入力列 10 を読むと状態 s_0，s_2 のいずれにも遷移できると解釈される．同様に s_0 で入力列 101 を読むと状態 s_0，s_1，s_3 のいずれにも遷移できるという解釈になる．これが，上で確認した $\hat{\delta}(s_0, 101) = \{s_0, s_1, s_3\}$ と上手く合致する．一つ注意をする．状態 s_1 で 1 を読んだ時の可能な次の状態は何であろうか．上の $\hat{\delta}$ の定義をすなおに適用すれば，これは存在しない，つまり，空集合が正解であることが分かる．行き先がないからといって現在の状態に留まるとは考えないでほしい．

入力列 x は $\hat{\delta}(s_0, x)$ が少なくとも 1 個の受理状態を含む時**受理される**と定義される．つまり，上の解釈を使うなら初期状態からテープ x を読んである受理状態に到達できるなら受理である．逆に如何なる受理状態にも到達できない時 x は受理されない．

ところでこの M_2 はどんな言語を認識するのであろうか．答えは，最後が 10100 で終わっている列の全体である．列が $x10100$，$x \in \{0, 1\}^*$ の形に書けるのであれば，初期状態から出発して，x を読んだ段階で明らかに s_0 に遷移できる．そこから 10100 でやはり明らかに s_5 に遷移できるので，この形の列は確かに受理される．逆に最後の 5 記号が 10100 でないなら，明らかに s_5 に到達することはできないことも分かる．

練習問題 2.7 言語 $\{x111 \mid x \in \{0, 1\}^*\}$ を認識する *nfa* と *dfa* を与えよ．

解答　nfa の方は上の M_2 の簡単な応用である．図 2.6 の状態遷移図で良いことは説明を要しないであろう（s_3 からの 1 よる遷移は無くても良い）．さて，決定性の方であるが，これは以

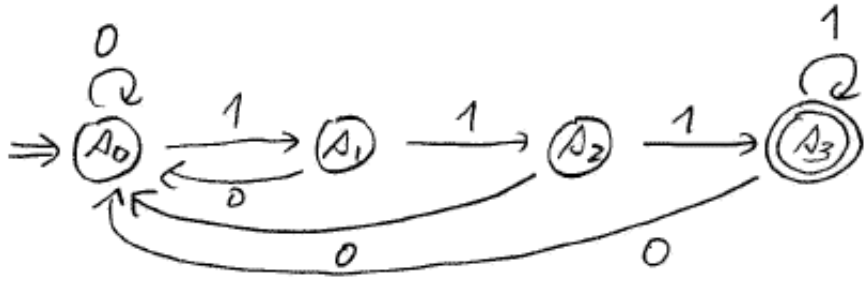

図 2.7 練習問題 2.6 の解答（決定性 fa）

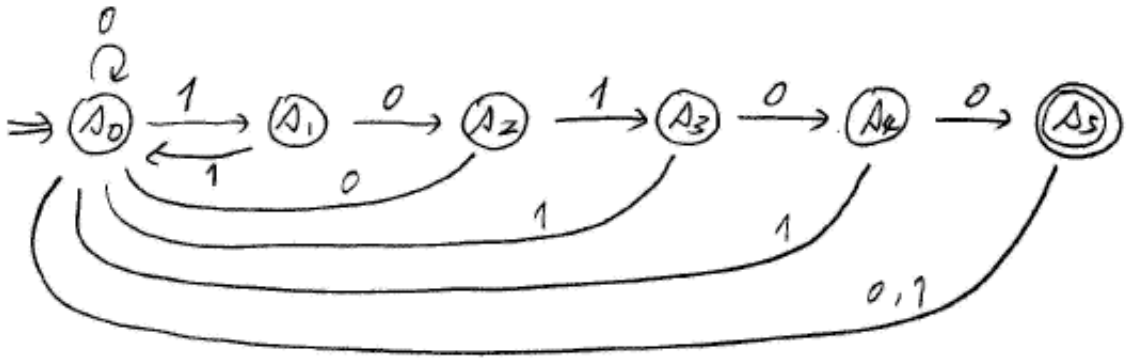

図 2.8 練習問題 2.7 の解答（不正解）

下の様に考えよう．まず，s_0 から 1 で自分自身に戻る遷移は取り除こう．つまり，1 を読めばそれが最後から 3 番目の 1 である可能性を考えて s_1 にのみ遷移するのである．さて，s_1 で 0 を読んだ時はどうすれば良いであろうか．これは簡単である．つまり，その前に読んだ 1 が最後から 3 番目の 1 である可能性が無くなってご破算になったのだから，s_0 に戻してやれば良いのである．同様に，s_2 や s_3 で 0 を読んだときも s_0 に戻してやる．こうして図 2.7 の様に決定性の有限オートマトンを得ることができた．（解答終り）

練習問題 2.8　以前の M_2 と同じ言語を認識する *dfa* を求めよ．

解答　練習問題 2.6 と同じだから簡単である．要するに，最後の 10100 であることを予想して遷移をして行って，思惑がはずれればまた最初からやり直せば良い．つまり図 2.8 の状態遷移図で良い．おっと，これは残念ながらちょっと不注意であった（実際テストにこの問題を出すとかなりの人がこの解答を書くのである）．例えば，列 1010100 を読んだときこの dfa は s_0 に遷移するので受理されない．しかし，この列は最後の 5 記号が 10100 であるから受理されなくてはいけないのである．正解は図 2.9 である．その正しさは各自が確認して欲しい．（解答終り）

練習問題 2.9　M_2 が認識する言語の補集合を認識する *nfa* を求めよ．

解答　M_2 で受理されない列はそれを読んだ後で非受理状態になる．補集合とはそう言う列を受理したいのであるから，以前の定理 2.2 と同様，図 2.5 の受理状態と非受理状態を入れ替えてやれば良い．

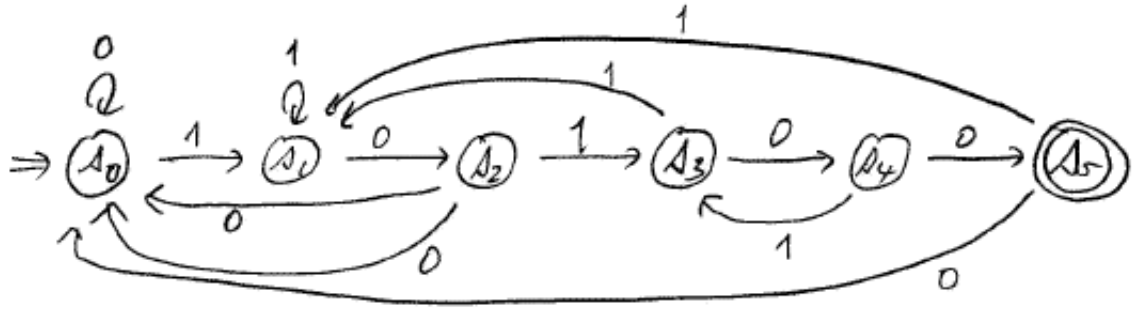

図 2.9 練習問題 2.7 の解答（正解）

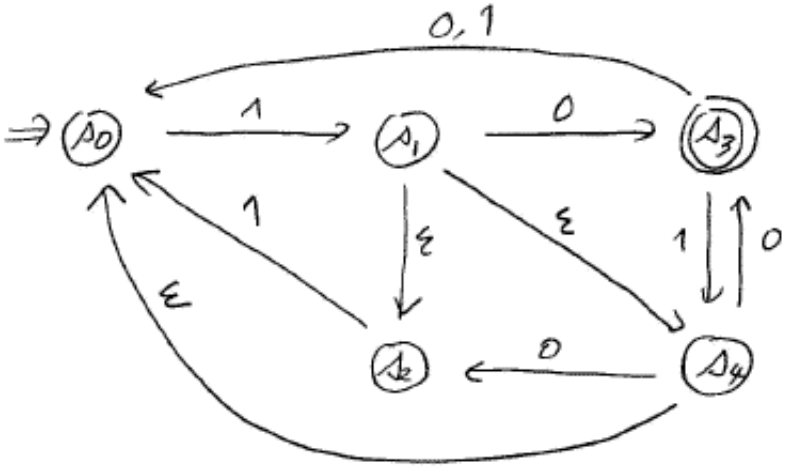

図 2.9-1 ϵ 入力付非決定性 fa

この解答も大間違いである．大体こんなことをやれば，図 2.5 の状態 s_0 が受理状態になってしまい，これは $\{0,1\}^*$ 上の列全部を受理してしまうことを意味する．決定性の時には正しかった議論が非決定性では通用しないことを良く理解して欲しい．勿論，図 2.9 のオートマトンは決定性なので，それに対して受理状態と非受理状態を入れ替えれば正解である．非決定性 fa を求めよと問題にはあるが，決定性 fa は非決定性 fa の特殊な場合であるから勿論決定性 fa で解答しても良い．　（解答終り）

我々は更に ϵ **入力付非決定性 fa**（ϵ**nfa**）と呼ばれるモデルを導入する．今までの fa の 1 ステップの動作は，入力を読んで，ヘッドを 1 コマ右に動かして，状態を変えるという 3 つから成っていた．ϵnfa ではこのうち，ヘッドを動かすという動作をしないことも許される．つまり，「入力記号を読まないで」状態遷移をすることもできる．例えば，図 2.9.1 の状態遷移図を見て欲しい．状態 s_1 からは 0 による遷移の他に，ϵ による遷移が状態 s_2 と s_4 に向けて定義されている．つまり，今オートマトンが s_1 にいるとすると，今のヘッドの下の記号が 0 であったなら状態 s_3 へ遷移してヘッドを 1 コマ右に動かすという通常の動作の他に，ヘッドを動かさないで s_2 あるいは s_4 に遷移するという動作も可能である．より形式的には状態遷移関数 δ が $K \times \Sigma \cup \{\epsilon\}$ から K の部分集合族（K の全ての部分集合の集合）の中への写像に変更になる．上の例なら，

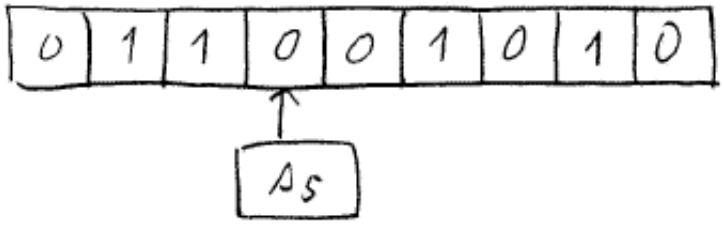

図 2.9-2 オートマトンの現在の状況

$\delta(s_1, 0) = \{s_3\}, \delta(s_1, \epsilon) = \{s_2, s_4\}$ といった感じになる．列の受理条件は以前と同じで，テープを読み終った（つまりテープの最後の記号を通常の動作で読んでヘッドを右に動かしてヘッドがテープからはずれた）時に，受理状態に到達できれば受理である．図 2.9.1 の ϵnfa は， 10 を受理することは今までと同じであるが，例えば， 1110 も受理する（$s_0, s_1, s_2, s_0, s_1, s_3$ の様に状態遷移して確かに受理状態に到達できる）．

このように列を受理する条件は直観的に分かっていただけたと思うが，やはり形式的に定義しないとまずい．ここでは，後でも便利になる**様相**という概念を利用して定義しよう．様相とはアルファベット Σ 上の列 x と状態 p に対し， (x, p) で与えられる．直観的には様相はオートマトンの動作途中の状況を表していると考えれば良い．例えば，オートマトンが図 2.9.2 の様な状況であるなら，それを様相 $(001010, s_5)$ で表す．ここで，つまり， x はテープのまだ読んでいない部分の列であり， p はそのときの状態である．既に読み終ったテープの部分を入れない理由は，将来の動作に影響を与えないからである．つまり，様相が与えられれば，その後の動作が定まる．

2 つの様相 $c_1 = (x_1, p_1)$ と $c_2 = (x_2, p_2)$ は以下の (i) 又は (ii) のいずれかの条件を満たすとき， c_1 から c_2 へ **1 ステップで移れる**と言い， $c_1 \Rightarrow c_2$ と書く． (i) $x_1 = x_2$ 且つ $p_2 \in \delta(p_1, \epsilon)$. (ii) ある記号 $a \in \Sigma$ に対して， $x_1 = ax_2$ 且つ $p_2 \in \delta(p_1, a)$. 図 2.9.1 のオートマトンなら，例えば， $(00011, s_1) \Rightarrow (0011, s_3)$ であり，また， $(00011, s_1) \Rightarrow (00011, s_4)$ である． $c_0 \Rightarrow c_1 \Rightarrow \cdots \Rightarrow c_k$ の時， c_0 から c_k へ**何ステップかで移れる**と言い， $c_0 \overset{*}{\Rightarrow} c_k$ と書く． 0 ステップ（つまり $c_0 = c_k$）も含む．列 x は，初期状態 s_0，ある受理状態 s_F に対し， $(x, s_0) \overset{*}{\Rightarrow} (\epsilon, s_F)$ である時**受理される**という．この定義によれば，入力テープを読み終った後で，更に何ステップかの ϵ 遷移を行って受理状態に到達する場合も含まれていることが分かる．

練習問題 2.10　図 *2.9-1* のオートマトンで列 110 が受理されることを示せ．

解答　$(110, s_0) \Rightarrow (10, s_1) \Rightarrow (10, s_4) \Rightarrow (10, s_0) \Rightarrow (0, s_1) \Rightarrow (\epsilon, s_2)$ であり，上の条件を満たしている．（解答終り）

2.4　有限オートマトンと正規表現の等価性

さて，これで準備が整ったので，懸案の（決定性） fa と正規表現の等価性を証明しよう．証明の手順は，まず， dfa と nfa， nfa と ϵnfa の等価性を証明する．なお， fa は特殊な nfa， nfa

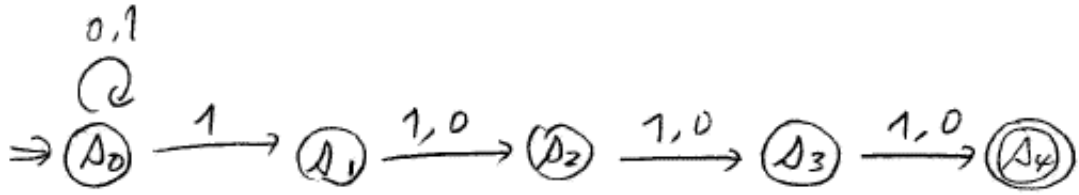

図 2.9-3 補題 2.1 の証明における nfa

は特殊な ϵnfa と考えることができるので，これらの性質はそれぞれ， nfa が fa でシミュレートできること， ϵnfa が nfa をシミュレートできることを示せば十分である．最後に， ϵnfa が正規表現でシミュレートできることならびにその逆を示す．

補題 2.1　言語 L が *nfa* で認識されるなら，ある *dfa* によっても認識される．

証明　与えられた nfa N に対し，それが認識する言語を変えずに dfa D に直せる（つまり，D は N をシミュレートする）ことを示せば良い．このために部分集合構成とよばれる良く知られた手法を用いる．アイデアは以下の通りである． N の状態集合の任意の部分集合を S_1 とする．初期状態からスタートした N がテープを途中まで読んだ時点で， S_1 の如何なる状態にも到達できると考えて頂きたい．すると，そこから更に記号 a を読んだ時に到達できる状態の集合 S_2 を以下の様に計算できる．

$$S_2 = \{q \mid q \in \delta(p, a)\text{である状態}p \in S_1\text{が存在する}\}$$

S_1 と S_2 を「単独の状態」とみなせば，これは S_1 から S_2 へ入力記号 a を読むことによって決定的に遷移するような fa の動作を模擬している，つまり我々が構成したい dfa D そのものであると考えることができる． N の部分集合を単独の状態とみなした M の状態を N の元の状態と区別するために s状態と呼ぶことにする．（図 2.9.3 の nfa を例にとると，例えば s状態 $\{s_0, s_1\}$ から記号 0 を読むと s状態 $\{s_0\, s_2\}$ へ遷移し， 1 を読むと s状態 $\{s_0, s_1, s_2\}$ へ遷移する．）

N の初期状態を s_0 とした時， s状態 $S_0 = \{s_0\}$ を新しい dfa D の初期状態とし，そこから記号 0 と 1 による遷移先の s状態を計算し，更にそれらの s状態からも 0 と 1 による遷移先を計算するという作業を続けていけば M の状態遷移図を決定できる． s状態が元の N の受理状態を 1 個でも含んでいれば新しい fa M の受理状態とする．この様な M が元の N と同じ言語を認識することは容易に想像できるので詳細は省く （証明終り）．

図 2.9.4 に図 2.9.3 の nfa を部分集合構成によって決定性に直した fa の状態遷移図の一部を示す．このように任意の非決定性 fa N は決定性の fa D に変換できるのであるが， D の状態数は N の状態数に比べて指数的に増加してしまう可能性があることがわかる．実際，図 2.9.4 の遷移図を完成させてみることをお進めする．そうすればこのことが良く分かる．逆に言えば，同じ言語を認識する fa でも非決定性で作れば状態数を大幅に節約できる可能性がある．このことは非決定性 fa の大きなメリットであることを記憶しておいて欲しい．

補題 2.2　言語 L が ϵ*nfa* E で認識されるなら， L はある *nfa* N によっても認識される．

証明　与えられた E が図 2.9.1 の遷移図だったとする．この E をシミュレートする nfa N （ϵ による遷移が存在しない）は図 2.10 の様に構成すればよい． N の状態は E の各状態に対応し

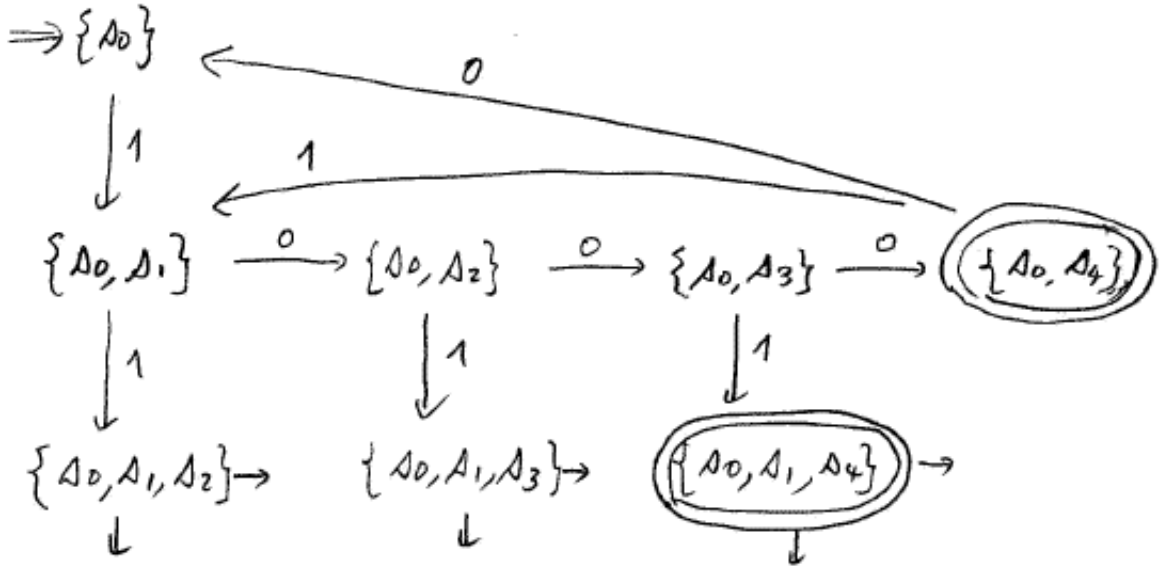

図 2.9-4 部分集合構成による決定性 fa の作成

たものを導入する．つまり，E の状態 s に対して，$(s, S(s))$ を対応させる．ここで，$S(s)$ は E の状態の集合で，s から入力記号を読まないで遷移できる状態を全て含む．例えば，E では s_0 からは ϵ による遷移は定義されていないので，この場合の $S(s)$ は s_0 のみを含む．s_1 からは s_1, s_2, s_4, s_0 に記号を読まないで遷移できる（s_0 へは s_4 を経由して）ので，対応する N の状態は $(s_1, \{s_1, s_2, s_4, s_0\})$ となる．なお，この状態からの 0 による遷移は $(s_2, \{s_2\})$ と $(s_3, \{s_3\})$ に向かって定義されている．その理由は $\{s_1, s_2, s_4, s_0\}$ の中の s_1 から s_3 へ，s_4 から s_2 と s_3 への 0 による遷移が E で定義されているからである．つまり，E において t_1 から t_2 へ $a \neq \epsilon$ による遷移が定義されているなら，$t_1 \in S(s)$ である様な $(s, S(s))$ から $(t_2, S(t_2))$ への遷移を設ける．このような N の構成が正当であることの証明は読者にお任せする．（証明終り）

さて，いよいよ正規表現と fa の関係に入ろう．最初に，任意の正規表現は ϵnfa でシミュレートできることを示す．既に，ϵnfa は nfa で，nfa は dfa でシミュレートできることを示しているので，結局，正規表現は dfa でシミュレートできることになる．次にその逆を示す，つまり dfa が正規表現でシミュレートできることを証明するのだが，実際には都合上 nfa を正規表現でシミュレートする．勿論これでも良い．

補題 2.3　言語 L が正規表現 R で表現されるなら，L はある ϵ*nfa* E によっても認識される．

証明　ある正規表現，例えば

$$R = (10^* + ((0 + 111)^* + 10)^* + 11^*)^*(000 + 111)^*100^*$$

が与えられたとしよう．結構ややっこしいので嫌気がさすかも知れないが我慢して欲しい．実際はこの何倍も長い式が与えられるかも知れないのである．しかしどんなに複雑な式が与えられたとしても，正規表現の定義によれば，その式は別の正規表現，R_1 と R_2 を使って，

$$R_1 + R_2,\ R_1R_2,\ R_1^*$$

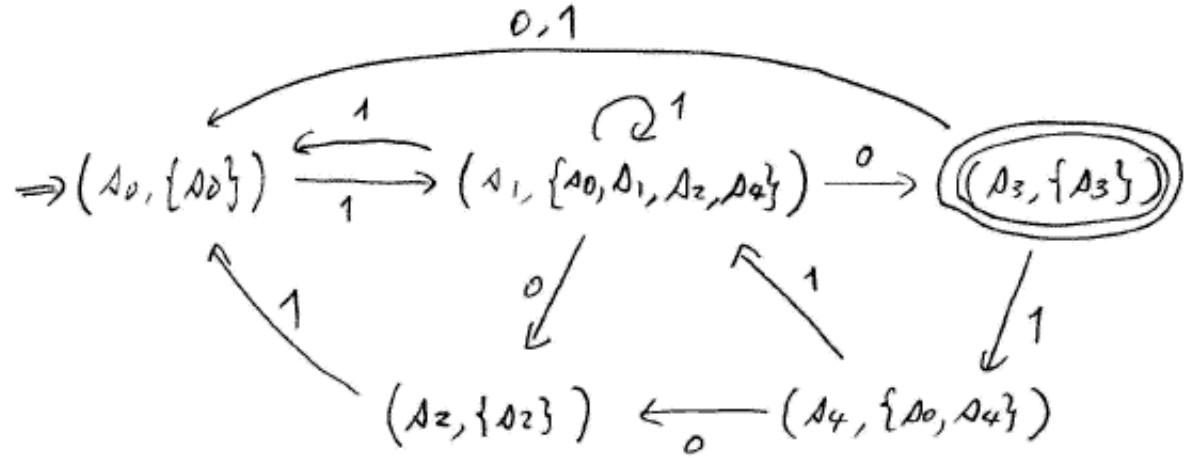

図 2.10 ϵnfa をシミュレートする nfa

のいずれかの形で書けるはずである．上の式の場合は良く観察すると，

$$R_1 = (10^* + ((0 + 111)^* + 10)^* + 11^*)^*,\ R_2 = (000 + 111)^*100^*$$

に対して，$R = R_1R_2$ と書ける．（この式ではカッコが省略されているので混乱するかもしれない．別の分解として，R_1 をもっと長くして，R_2 の $(000 + 111)^*$ まで含めてしまうことも可能である．連接演算は結合律が成立するのでいずれでも構わない．しかし，以下では与えられた正規表現はカッコが省略されていないと考えた方が楽である．このような分解の多様性が無くなって，一意に決まるからである．）次に R_1 と R_2 のそれぞれを更に分解していくことができる．例えば，R_1 の場合は $R_3 = 10^* + ((0 + 111)^* + 10)^* + 11^*$ と置いて，$R_1 = (R_3)^*$ と書ける．このことをどんどん続けて行くと，新たに導入される正規表現 $R_4, R_5, \cdots$ はどんどん簡単になっていくことが分かる．最終的には，例えば $R_{56} = 0 + 1$ となって，それは $R_{57} = 0$ と $R_{58} = 1$ を使って，$R_{56} = R_{57} + R_{58}$ と書け，この時点で R_{57} と R_{58} はもはやより簡単な正規表現に分解できない．もうこれ以上分解できない正規表現としては，定義によって，0 や 1 の様な 1 個の記号，ϵ，ϕ の 3 種類の可能性があることを思い出して欲しい．

このことが分かれば証明はほぼ終ったも同然なのである．我々は上の分解に沿って fa を構成していく．式を使っても良いのであるが，より分かりやすいので，状態遷移図を使って説明しよう．最初に正規表現 R が与えられたら，図 2.11 の様 (a) の様に，1 個の初期状態 s と 1 個の受理状態 t だけの「オートマトンの様なもの」を用意する．s から t へは正規表現 R による遷移枝を設ける．気持としては $L(R)$ に入る列を読んだときに s から t へ遷移すると読んでほしい．その後は，この 2 状態と 1 本の遷移を R の分解に従って置き換えて行けばよい．それを同図の (b)，(c)，(d) に示した．(b) は分解が $R = R_1R_2$ の場合である．新たな状態を 2 個用意して遷移枝の部分を置き換えたと考えれば良い．(c)，(d) の場合も同様である．ここで，更に R_1 の遷移枝を置き換えてという様にどんどん置き換えを続けていけば良い．ここでちょっと注意して欲しいのは，置き換える部分は常に，左の状態から右の状態へ，ある正規表現の枝が 1 本だけ出ているという状態が保たれている．これは ϵ による遷移を上手に利用した結果であって，ϵ による遷移がもし使えないとすると（特に (c) が）結構面倒になる．

この枝の置き換えをそれ以上できなくなるまで続けて行くと，各枝に付いている正規表現は，

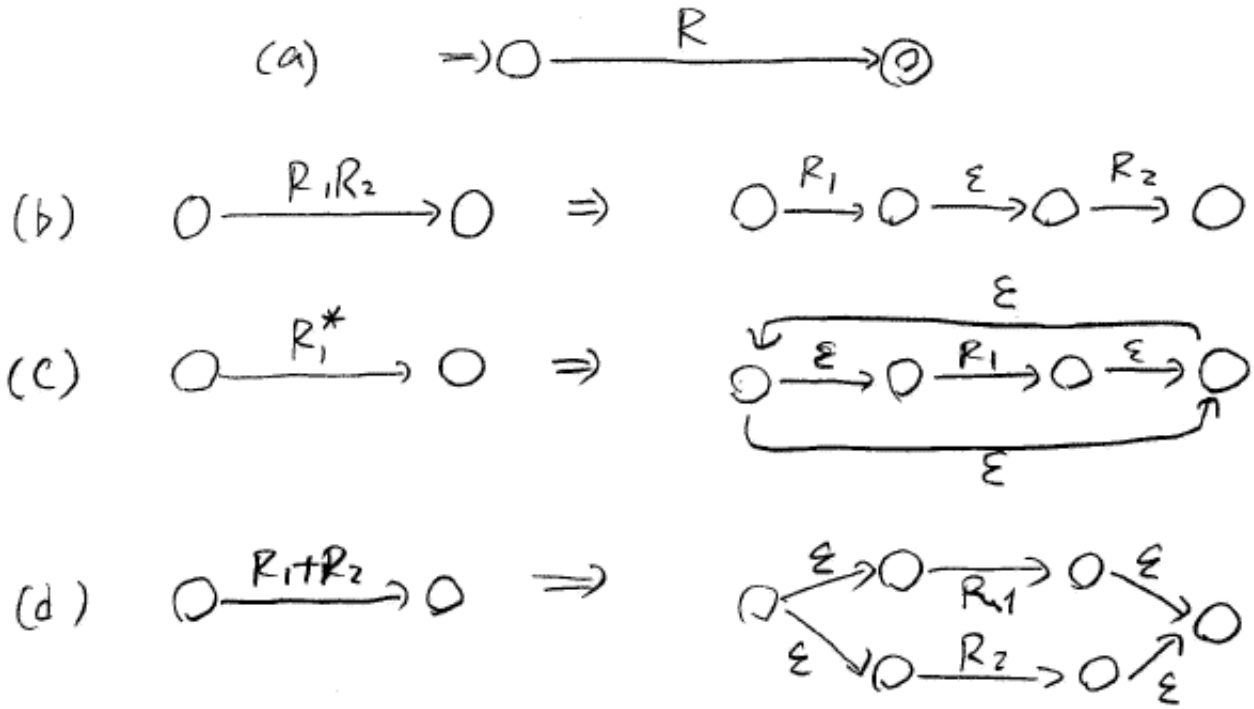

図 2.11 正規表現の有限オートマトンによる模倣

1 個の記号，ϵ，ϕ のいずれかになる．全 2 者は ϵnfa で許されているものなのでそのままで良い．ϕ の場合は単にその枝を取り去ってしまう（その状態からの，ϵ，0，1 による遷移先を全て空集合にする）．これで ϵnfa が完成する．

以上述べた構成が正しいことを証明しなければならないが，それはほとんど明らかであろう．枝に正規表現 R_i が付いているときに $L(R_i)$ に属す列を読んでその枝を遷移すると考えればよい．そうすれば，上の 3 個の変換が状態 s から状態 t への遷移できる言語を変えていないことが分かるであろう．例えば，図 2.11 の (d) の場合は，変更前も変更後も $L(R_1)$ か又は $L(R_2)$ に属す列を読んで s から状態 t に遷移することを意味している．従って，変更の全過程において，最初の (a) で約束された，「初期状態から受理状態へ与えられた正規表現 R で表現される言語に属する列を読んで遷移する」という事実が保存されていることになる．（証明終り）

練習問題 2.11　正規表現 $0^*(1+\phi 0^*)^*$ が表現する言語を認識する *ϵnfa* を上の手順で構成せよ．

解答　最終的な答のみを図 2.12 に与える．中間のプロセスを是非実行してみて欲しい．（解答終り）

補題 2.4　言語 L が *nfa* N で認識されるなら，L はある正規表現 R によっても表現される．

証明　N の状態遷移図を書こう．そして，どれでも良いから 1 本の遷移枝に注目する．この状況を図 2.13 に与える．今我々は状態 s から s' への記号 a による遷移に注目している．さて，ここで我々はこの 1 本の遷移を取り除いた nfa を考えそれを M と置く．「取り除く」とはいい

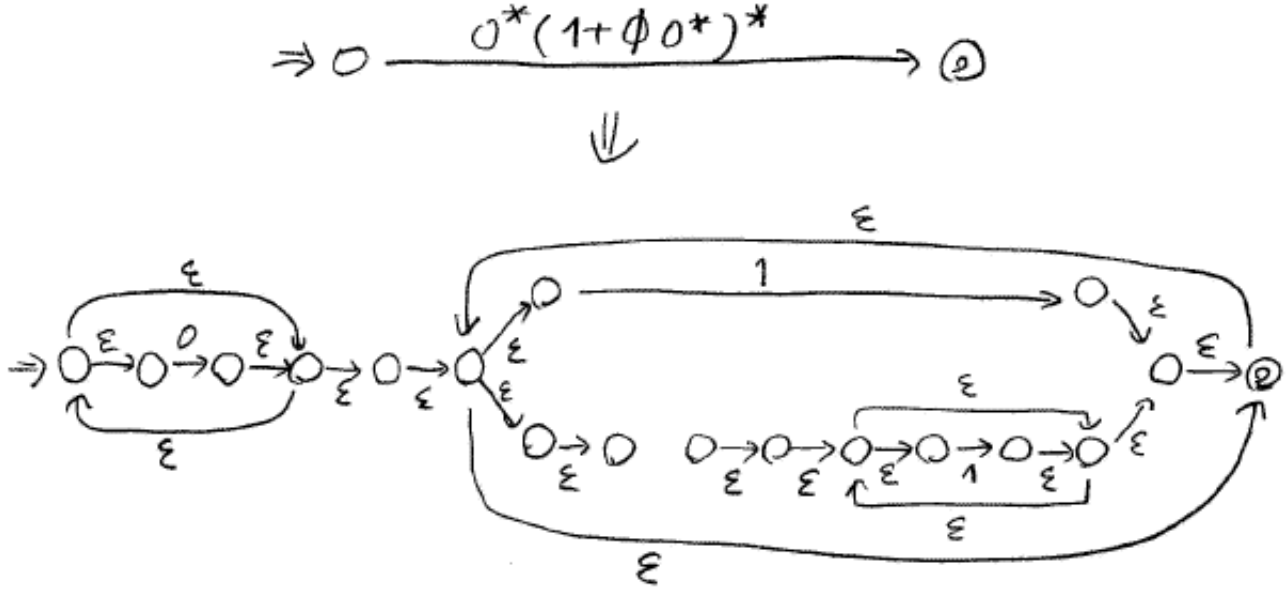

図 2.12 正規表現から fa への変換の例

加減な表現で恐縮だが，要するに状態遷移関数の

$$\delta(s,a) = \{\cdots, s', \cdots\}$$

の部分の右辺を s' を取り除いた集合にすることである．（もし s' だけしか無かったら空集合にすれば良い．）更にこの M の初期状態をある状態 v に，また受理状態集合をある部分集合 V に変更した fa を $M(v,V)$ で表す．例えば $N(s_0,\{s\})$ は初期状態は元の N と同じ，受理状態が 1 個（s）だけの nfa である． F は元の N の受理状態集合を表す．更に， nfa X が認識する言語を表現する正規表現を $Reg(X)$ で表すことにする．例えば図 2.9-3 をもう一回見て欲しい． s_1 から s_2 へは 0 と 1 の両方の枝があるが，もし 0 の方の枝を取り除いてしまうと，このオートマトンは最後から 4 番目と 3 番目の記号が共に 1 である列全体を受理することになる．また s_0 から s_1 への枝を取り除いてしまうともはやいかなる列も受理しない．つまり空集合を認識するオートマトンになってしまう．

さて，図 2,13 に戻ろう．この証明の核心は以下の関係を理解する事である．

$$Reg(N) = Reg(M(s_0,F)) + Reg(M(s_0,\{s\}))[a \cdot Reg(M(s',\{s\}))]^* \cdot a \cdot Reg(M(s',F))$$

もし見にくいなら $A = Reg(M(s_0,F))$， $B = Reg(M(s_0,\{s\}))$， $C = Reg(M(s',\{s\}))$， $D = Reg(M(s',F))$ とおいて書き直すと，

$$Reg(N) = A + B(aC)^*aD$$

となる．この関係が正しいことは以下のようにして分かる．まず， nfa N がある列 x を受理するとはどういうことであろうか．それは，勿論，初期状態から x を読んである受理状態 s_F に到達できることである．この到達の道筋は遷移枝を並べることによって書くことができる．（再び

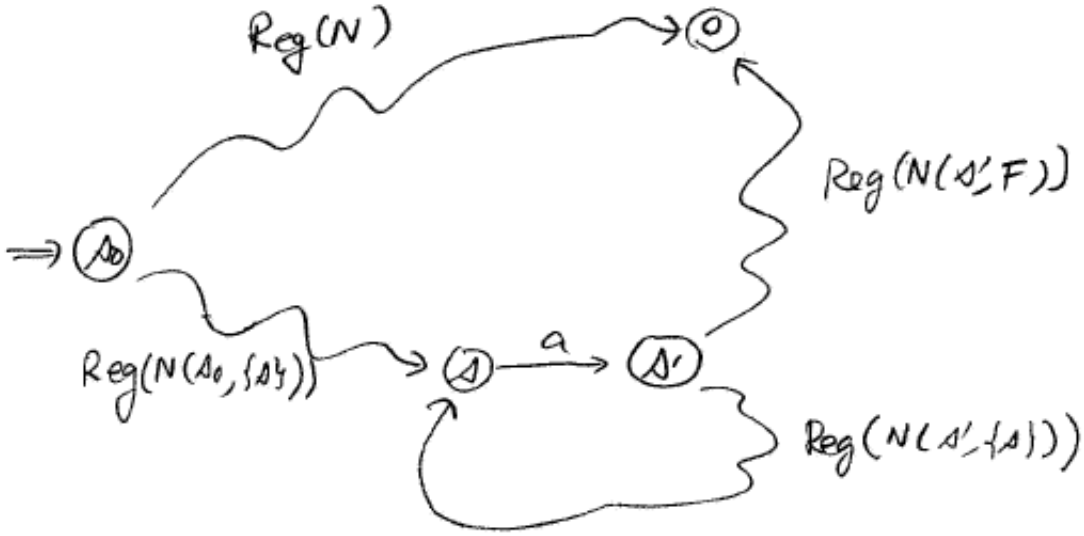

図 2.13 与えられた fa の遷移図

図 2.9-3 を見るなら， 001100 は受理される．その道筋は s_0 から s_0 への 0 の枝， s_0 から s_0 への 0 の枝， s_0 から s_1 への 1 の枝， s_1 から s_2 への 1 の枝， s_2 から s_3 への 0 の枝， s_3 から s_4 への 0 の枝，の様に遷移枝の列で書ける．）

x を読んで N の s_0 から s_F に到達する道筋に M では取り除かれている枝（これを枝 a と呼ぶことにしよう）が含まれているかどうかが問題になってくる．もし，含まれていないなら，その様な x は $M(s_0, F)$ でも受理されることが分かる．これは上の正規表現でいうなら， x は $A = Reg(M(s_0, F))$ で表現される言語に入る．次に，道筋に枝 a が 1 回現れた場合を考えよう．この場合は，道筋は s_0 から（枝 a を通らずに）まず s に到達し，枝 a を通って s' へ行き，そこから s_F に行くので，上の正規表現でいうなら， BaD という表現の言語に入っているはずである．同様に考えるなら，枝 a を 2 回通る場合は正規表現 $BaCaD$ が対応し， 3 回なら $BaCaCaD$ が対応する．従って，どのような x であったとしても，上の関係式の右辺の正規表現 $A+B(aC)^*aD$ でカバーされているこが分かる．

さて，次は上の関係式の右辺の $Reg(M(s_0, F))$ に着目して， N より 1 本枝の少ない fa $M(s_0, F)$ の枝を更に 1 本除去したオートマトンを作る． $Reg(M(s_0, F))$ はその新しいオートマトンを使った式で置き換えられることになる．どんどんこれを繰り返して行くと，右辺の式に含まれるオートマトンの枝数（元々有限であることに注意）はどんどん減少して行って，やがては 0 本になってしまう．遷移枝の全く存在しないオートマトンとは奇異な感じがするかもしれないが nfa の定義からは全く逸脱していない．そのような nfa を Y とすると， $Reg(Y)$ は Y の初期状態が受理状態の時は ϵ，受理状態で無いときは ϕ で置き換えられることが容易に分かる．この時点で，上の関係式の右辺は完全な正規表現になった．

こうして作成された正規表現が与えられたオートマトンと等価である（同じ言語を表現する）ことの証明は読者にお任せすることにしよう．これは，上の説明からほぼ明らかであろうが，形式的に行おうとすると結構やっかいである．(証明終り)

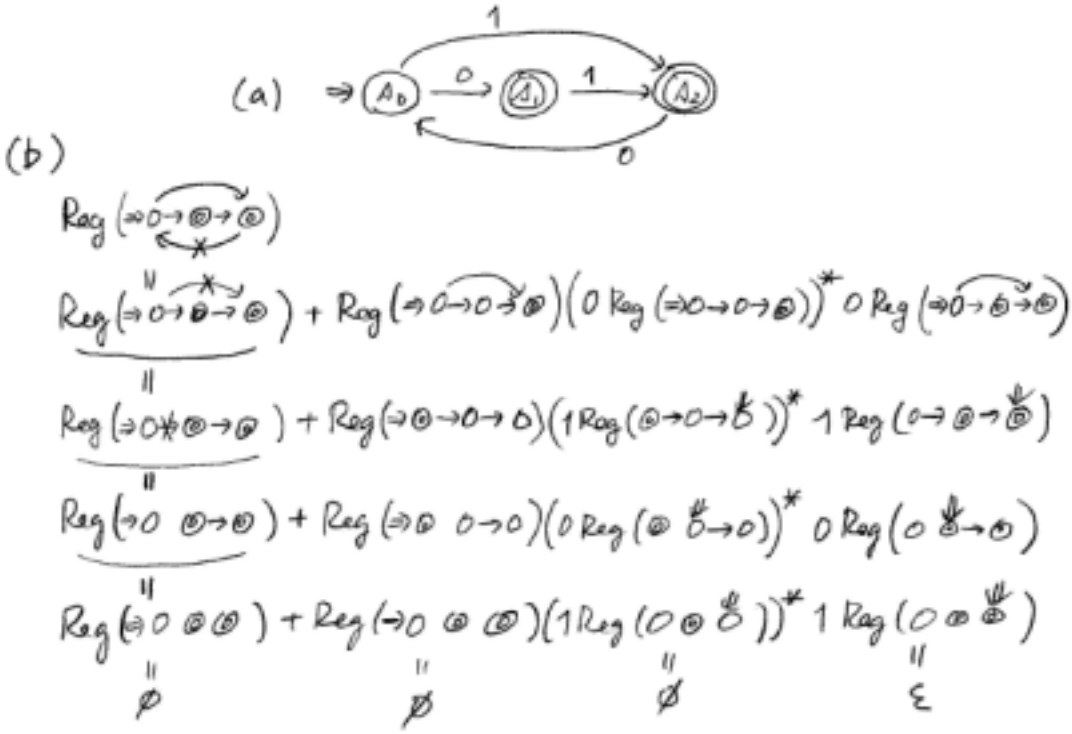

図 2.14 fa から正規表現への変換の例

練習問題 2.12　図 *2.14(a)* に与えた *nfa* に対してそれと等価な正規表現を上の手順に従って構成せよ．

解答　同図 (b) にその一部を示す．この図の読み方は特に説明を加えないが明らかであろう．（解答終り）

以上 4 個の補題を総合して，以下の定理を得る．

定理 2.5　*fa* と正規表現は等価である．

2.5　有限オートマトンの能力の限界

有限オートマトンならびに正規表現は簡潔で分かりやすく，また実用性も高く，言語を表現するための組織的手法としては極めて良くできている．例えば，前者は文字列を表現するための道具として UNIX の重要な一部分になっているし，後者はいわゆる状態遷移図という形で様々な場所で便利に利用されている．更には，前説で証明した fa と正規表現の等価性は，言語・オートマトン理論における最初の非自明な結果として有名で，今でもその美しさを失っていない．このように，有限オートマトン・正規表現の重要さは多くの人の認めるところであるが，その言語表現能力という観点からみると，残念ながら高くない．

有限オートマトンの能力が高くない原因ははっきりしている．状態が有限個しか存在せず，その範囲内でしか記憶ができないからである．例えば，前章で現れた左カッコと右カッコがバランス

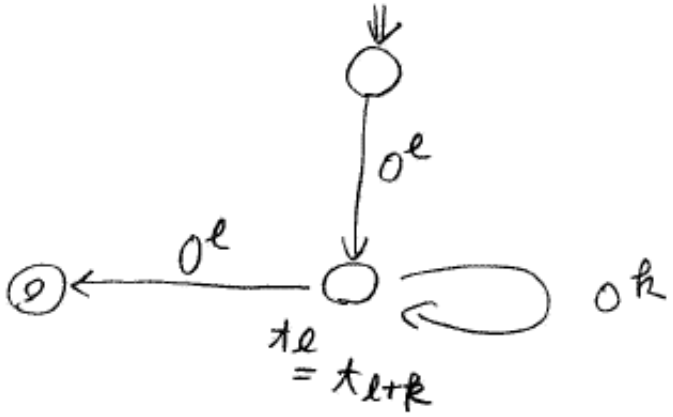

図 2.15 鳩ノ巣原理の応用

した列を考えて見よう．

$$((((((((()))))))))$$

の様な列である．この列の正しさを検証しようとするなら，左カッコを読み終った段階で，その個数（上の例なら 8 個）を何らかの方法で記憶しておかないと右カッコとの比較ができない．しかし，上で述べた様に fa の記憶は「有限の状態」に頼るしかない．従って，例えば状態数が 10 しかない fa なら， 11 個以上の左カッコが与えられた時にはもはや対処できない．このように考えると，言語

$$\{0^n 1^n \mid n \geq 0\}$$

が fa で認識できないのは明らかである．しかし，数学的に証明しようとすると，上のような直観的説明では勿論不十分である．

定理 2.6　言語 $\{0^n 1^n \mid n \geq 0\}$ はいかなる *fa* によっても認識できない．

証明　1 章で説明した鳩ノ巣原理が便利である．上の言語がある fa M で認識できたと仮定する． M の状態数を n とする．さて， M の初期状態から記号 0 による遷移をどんどんたどって行って見よう．すなわち，

$$t_i = \delta(s_0, 0^i)$$

と置く．状態数は n なので， $t_0, t_1, \cdots, t_{n+1}$ が全て異なった状態になることは不可能で，これらの中には少なくとも 2 個の同じ状態が現れる．それらを， t_l と t_{l+k} （$k \neq 0$) としよう（図 2.15 参照）．列 $0^l 1^l$ は言語に入っているので受理されなくてはならない．つまり， $\delta(t_l, 1^l)$ は受理状態である．しかるに，このことは，列 $0^{l+k} 1^l$ も受理してしまうことを意味しており， M が言語 $\{0^n 1^n \mid n \geq 0\}$ を認識することに矛盾する．（証明終り）

練習問題 2.13　言語 $\{xx \mid x \in \{0,1\}^*\}$ はいかなる *fa* によっても認識されないことを示せ．

解答　これは実際試験問題に出したことがある．以下の様な解答が（複数）見られた．問題の言語を認識する fa が存在したと仮定して， M と置く．その状態数を n とする．任意の長さ $n+1$ の列 $a_1 a_2 \cdots a_{n+1}$ を選んで，その列を読んだときの M 初期状態からの遷移を観察する．上の定理の証明と同じ理由で，必ず同じ状態を 2 度訪れることになる．いま， $a_1 \cdots a_i$ を読んだ後の

状態と $a_1 \cdots a_i a_{i+1} \cdots a_{i+j}$ を読んだ後の状態が一致したとする． $a_1 \cdots a_i a_1 \cdots a_i$ は受理されるべき列なので，このことは列 $a_1 \cdots a_i a_{i+1} \cdots a_{i+j} a_1 \cdots a_i$ も受理してしまうことを意味する．これは矛盾である．

この解答は残念ながら明らかに間違いである．列 $a_1 \cdots a_i a_{i+1} \cdots a_{i+j} a_1 \cdots a_i$ も受理されてしまうから矛盾だと言っているが，どこが矛盾しているのであろうか．まあ，確かに列 $a_1 \cdots a_i a_{i+1} \cdots a_{i+j} a_1 \cdots a$ は何となく xx の形をしていない様に見えるかもしれないが，良く考えてみれば，例えば， $a_1 \cdots a_i = 100100100$， $a_{i+1} \cdots a_{i+j} = 100100$ であったらいかがであろうか．列 $a_1 \cdots a_i a_{i+1} \cdots a_{i+j} a_1 \cdots a_i$ は受理されて良いのである！

上の解答の誤りは，不注意にも「任意の列を選んで」しまったことによる．任意の列では上のような矛盾を生じさせないような列もありうるわけで，証明としては全く 0 点である．そのような不都合が生じないような列を選べば良いわけで，例えば，列 10^n などはいかがであろうか．この列を使えば上の定理とほぼ同様の証明ができる． （解答終り）

2.6 まとめと文献

この章に現れた内容は 1950 年代に既に分かっていたことである．但しその当時の論文を調べてみると，その「書き方」が現代とはかなり異なっていていささか読むのに苦労する．それは当然と言えば当然で，その当時このような話題は全く新しい話題で，研究者も物理学や数学から転身した（完全では無くて片手間の人も多かったに違いない）人だった，そのような研究者が自分達の古巣の流儀を持ち込んだわけである．パイオニア的論文が数件あるのであるが，それらの紹介は入門書である本書では避けて，他のより専門的な本にお任せする．ただ，以下の 2 つの文献は是非紹介したい．

まず，日本の研究者が深く関わっている論文をあげさせて頂く．元東大の山田尚勇先生はこの分野の草分の一人で，オートマトン理論，計算量理論でいくつかの重要な仕事をされている．以下の論文は本書でも論じた正規表現と fa の関係等を論じている．本書での fa から正規表現，又その逆の変換は極力分かりやすさを重視したものになっているが，以下ではより効率の良い変換を与えている．

・ R. McNaughton and H. Yamada, “Regular expressions and state graphs for automata,” IRE Trans. Electronic Computers, 9, 1, 39-47, 1960.

また，以下の論文は有限オートマトンの論文とは言えない面もあるが，この分野の論文の「スタイル」を作ったという意味で大変重要な論文である．著者が学生であったころも最初に読まされた論文である．上で，書き方が違っていて読みづらいと言ったが，この論文に関する限りそれが無い． 60 年代に現れた膨大な数のオートマトン理論の論文では，議論の進め方や取り扱う問題がほとんどこの論文に習っているというと言っても過言でない．なお， Rabin は（勿論他にも重要な仕事を数多く発表して）情報・計算機の分野で最も権威が高いと言われているチューリング賞を受賞したが，この初期の仕事も受賞理由に入っている．

・ M. Rabin and D. Scott, “Finite automata and their decision problems,” IBM J. Res. and Develop., 3, 2, 115-125, 1959.

最後に補題 2.1 で述べた非決定性有限オートマトンの決定性有限オートマトンによる模倣の話をちょっと補強する．元の非決定性 fa の状態数が n の時に決定性 fa では 2^n 個の状態を使用すれば必ずできるという状態数の上限は古くから知られていたが，実際にどの程度の状態数が必要になるかという下限の研究はほとんど行なわれていなかった．最近著者のグループがこの下限に関して，ほとんどの $k \geq 0$ に対して，実際に決定性では $2^n - k$ 状態必要になる例が存在することを証明した．

・K. Iwama, Y. Kambayashi, and K. Takaki, "Tight Bounds on the Number of States of DFA's That Are Equivalent to n-State NFA's" Theoretical Computer Science 237, pp. 485-494 (2000).

・K. Iwama, A. Matsuura, and M. Paterson, "A Family of NFA's which Need $2^n - \alpha$ Deterministic States," Proc. MFCS 2000, pp. 202-209 (2000).

理解度の確認

問　練習問題 2.2 を数学的帰納法を用いて厳密に証明せよ．

問　図 2.9-3 は最後から 4 番目の記号が 1 である $\{0,1\}$ 上の列の全体を受理する nfa であるが，この言語を一般化すると最後から k 番目の記号が 1 である列の全体になる．この言語を受理する fa を非決定性でなら $k+1$ 状態で作れることは容易に分かる．決定性の fa に直そうとすると，状態数が 2^k くらいまで増加してしまうことが図 2.9 から推察されるであろう．このことを証明せよ．（ヒント．簡単のため $k = 5$ としよう．状態数が 2^5 より少ない dfa に長さ 5 の 0 と 1 の列を読ませた場合を考えよう．列の種類は 2^5 であるから，少なくとも 2 つの異なった列に対して，読み終った時の状態が同じになる．このような列を例えば，10111 と 10011 としよう．良く見ると左から 3 番目の記号が 1 と 0 で異なっている．さて，これらの列の後に 00 を付加して見よう．1011100 は右から 5 番目の記号が 1 であるから受理される．そうすると 1001100 も当然受理される．しかし，これはおかしい．）

問　補題 2.2 をより形式的に証明せよ．（方針．図 2.9-1 の様な ϵnfa がある列を受理するということは，その列を読んで初期状態から受理状態へ遷移するような道筋があるということである．例えば

$$s_0 \xrightarrow{1} s_1 \xrightarrow{\epsilon} s_4 \xrightarrow{\epsilon} s_0 \xrightarrow{1} s_1 \xrightarrow{0} s_3.$$

の様な遷移の列である．これに対して，図 2.10 では

$$(s_0, \{s_0\}) \xrightarrow{1} (s_1, \{s_1, s_2, s_4, s_0\}) \xrightarrow{1} (s_1, \{s_1, s_2, s_4, s_0\}) \xrightarrow{0} (s_3, \{s_3\}).$$

という遷移の列が対応する．つまり，前者で受理される列は後者でも受理される．逆も同様にして証明される．）

問　練習問題 2.5 の言語 L_1 を多少変更し，さらに一般化して以下の言語 L_k とする．$L_k = \{x | x \in \{0,1\}^*, \sharp_0(x) = \sharp_1(x),$ 且つ任意の x のプレフィックス y に対して，$0 \leq |\sharp_1(y) - \sharp_0(y)| \leq k\}$．$k$ を正定数としたとき，L_k を正規表現で表せ．（ヒント．練習問題 2.5 でやったように fa で表現する方が明らかに易しい．従って，まず L_k を認識する fa を構成してそれを補題 2.4 の方法で正規表現に直すという方法も原理的には可能であるが，かなり大変である．やはり直接正規表現を

導くことにしよう．まず，L_1 を考えよう．これは簡単で $(10+01)^*$ で良いであろう．次に L_2 であるが，これは以下のような考えがヒントになる．例えば列

$$1101001100$$

は L_2 に属す列であるが，最初の 1 と最後の 0 を取り除くと，L_1 に属す列になっている．）

問　2 つの fa M_1 と M_2 が等しい言語を認識するかどうか（M_1 と M_2 が等価かどうか）を判定する方法を述べよ．

問　言語 $\{0^p \mid p\text{は素数}\}$ は fa では認識できないことを証明せよ．

問　言語 L がある fa で認識されるとする．この時，言語 $XX(L) = \{x \mid xx \in L\}$ もまたある fa によって認識されることを証明せよ．（ヒント．まず，練習問題 2.10 の言語と混同してはいけない．L を認識する fa を M とする．列 x が $XX(L)$ に入っているということは，M が初期状態 s_0 から x を読んで，ある状態 s に到達したなら，その s から更に x を読んである受理状態（$\in F$）に達するということを意味する．つまり，そのような列は $M(s_0, \{s\})$ が受理する言語と $M(s, F)$ がが受理する言語の共通集合である．途中の状態はどれでも良いので，このような集合の和集合を取れば良い．）

問　fa M_1 と M_2 が認識する言語の共通集合を認識する fa の作り方を定理 2.2 で示したが，その状態数の上限に関して論じよ．その様な上限の状態数が実際に必要になる例を与えること．（ヒント．一つ前の問題のような 1 文字の言語が分かりやすい．）

問　ϵ 遷移は使っても良いが，見掛け上は決定性の fa を考えよう．つまり，状態遷移関数が $K \times (\Sigma \cup \{\epsilon\})$ から単に K の中への写像となっているようなものである．このような fa で非決定性の fa を状態をあまり増やさずにシミュレートする方法を考えよ．

第 3 章

文脈自由文法

前章で言語を表現するための手法として，fa と正規表現を導入して，「大変良くできたシステムであるが能力が低い」ことを言った．そこで，より能力が高いシステムを導入するわけであるが，それが本章と次章で導入する文脈自由文法とプッシュダウンオートマトンである．以下ではまずなぜ文脈自由文法とプッシュダウンオートマトンが重要であるかを述べた後，文脈自由文法の定義といくつかの例を与える．文脈自由文法は多くの重要な性質を持っていてそれだけでも立派な 1 冊の本になるが，それらについては後で必要になる最小限のものにとどめる．最後に文脈自由文法の能力の限界について言及するが，その証明は fa の場合に比べて格段に困難である．

3.1 文脈自由文法の重要性

文脈自由文法と次章で述べるプッシュダウンオートマトンはいくつかの理由で極めて重要である．

第 1 の理由として，通常形式言語の世界で「文法」といえば文脈自由文法を意味するくらい，文脈自由文法は文法の典型的クラスとなっている（クラスという言葉を説明なしで使ってしまったが，とりあえずはグループと言った感じでつかんでおいて欲しい）．ということは，他の文法も存在するわけで，例えば有限オートマトンに対応する文法（正規文法と呼ばれる）も存在する．しかし，それは文法の能力を十分使っているとは言えず，あたかもベンツを時速 40 キロで運転するようなもので，ミスマッチの感が否めない．一方で，より能力の高い文法も存在するが，こんどは逆に数学的定義を駆使して無理に能力を高めた印象でやはりピッタリと来ない．

第 2 の理由として，文脈自由文法で表現できる言語（文脈自由言語と呼ばれる）の重要性である．多くのプログラミング言語がこのクラスに入っていることが分かっている．言い過ぎかも知れないが，様々な目的で人工的な言語を開発する時には，ほぼそれは必然的に文脈自由言語になると言っても良いのではなかろうか．

第 3 の理由として，文脈自由文法とプッシュダウンオートマトンの関係である．前章で正規表現と fa の等価性が非自明で面白いことを学んだが，両者の関係（やはり等価である）はそれにも増して，見事なくらい巧妙である．その証明は次章に回すことになるが，初学者が結構理解す

るのに苦労する話題の一つになっている．本書ではそれほど困難無くその巧妙さを味わって頂けるよう努力をつぎ込むので，どうか楽しみにしていて欲しい．

第 4 の理由は少し今までの理由とは毛色が異なる．すなわち，文脈自由文法は我々が持っている「文法」のイメージと非常に良く合致したシステムである．例えば，英文法で，文 → 名詞句・動詞句，名詞句 → 前置詞・名詞，動詞句 → 動詞・形容詞句，といった構造の説明をおぼえておられるのではないだろうか．前置詞の例が the，名詞の例が dog，動詞の例が is，形容詞句の例が cute で，それらを使って置き換えてみると， the dog is cute という文章ができるのである．以下の説明でも分かるように，文脈自由文法の構造も非常に良く似ている．従って，文脈自由文法を知ることによって，逆に「ああ，文法とはこういうものだったのか」という理解が得られるのである．

3.2 文脈自由文法

文脈自由文法は，どちらかと言えば正規表現に近い．すなわち，言語に属する列を「生成する」システムである．

文脈自由文法（cfg）は $G = (V, \Sigma, P, S)$ で与えられる．ここで， V と Σ は有限集合で，要素はそれぞれ**変数**，**終端記号**と呼ばれる． P は**生成規則**の有限集合で，一つの生成規則は変数 A と，列 $\alpha \in (V \cup \Sigma)^*$ に対して， $A \to \alpha$ の形をしている． S （$\in V$）は特別な変数で，**開始記号**と呼ばれる．

例えば， $G_1 = (\{S\}, \{0, 1\}, \{S \to \epsilon, S \to 0S1\}, S)$ は cfg である．さて，この文法はどんな言語を表現するのであろうか．形式的定義は後回しにして，より直観的に説明する． G_1 は以下のような導出という操作によって， G_1 が表現する言語（$L(G_1)$ と書かれる）に属する列を導く．

$$S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 000S111 \Rightarrow 000111.$$

つまり，導出とは，開始記号から始めて，生成規則をどんどん適用することによって行われる．ここで，「適用する」とは，列の中に現れる変数を生成規則の右辺で置き換えることである．例えば，上の導出の例で， $00S11$ に現れる変数 S を生成規則 $S \to 0S1$ の右辺 $0S1$ で置き換えることによって， $000S111$ が得られる．これが導出の 1 ステップである．何ステップかの導出によって，変数を含まない終端記号のみからなる列が導ければその列は $L(G_1)$ に入っている．例えば，上の導出が存在するので， $000111 \in L(G_1)$ である．同様に 00001111 等も導出できることが容易に分かる．結局 $L(G_1) = \{0^n 1^n \mid n \geq 0\}$ である．

注意してほしいのは S を生成規則の右辺で置き換えるときに，適用できる生成規則（つまり，左辺が S の生成規則）が 2 つ存在することである．どちらを適用してもよいので，ここに自由度が生じ，数多くの列を導くことができる．上の例では各ステップで，そこまでに導出された列には変数が 1 個（S）しか現れていない．しかし，一般には複数個現れるので，どの変数に対して生成規則を適用するかという自由度もある．それでは，導出の形式的定義を与えよう．

2 つの列， $u, v \in (V \cup \Sigma)^*$ が次の条件を満たすとき， v は u から **1 ステップで導出される**と言い， $u \Rightarrow v$ と書く．条件：ある $\alpha, \gamma \in (V \cup \Sigma)^*$ とある生成規則 $A \to \beta \in P$ が存在して，

$u = \alpha A \gamma$，且つ $v = \alpha\beta\gamma$ と書ける． v が u から何ステップかで導出されるとき単に**導出される**と言い， $u \overset{*}{\Rightarrow} v$ と書く．以前の様相の時と同じように 0 ステップも含むので， $u = v$ なら常に $u \overset{*}{\Rightarrow} v$ である．

文法 G の開始記号を S とした時，変数を含まない列 $x \in \Sigma^*$ が S から導出されるとき， x は文法 G によって，**生成される**という．生成される列の全体が文法 G が**生成する言語**であり， $L(G)$ と書かれる．言語 L が文脈自由文法によって生成されるなら L は文脈自由言語と呼ばれる．

練習問題 3.1　言語 $\{0^n 1^m \mid n \neq m\}$ を生成する *cfg* を求めよ．

解答　上で見た文法 G_1 は $n \neq m$ ではなくて， $n = m$ であるような列を生成する．これを利用しない手はない．答えは $G_2 = (\{S, S', A, B\}, \{0, 1\}, P, S)$ である．ここで，生成規則の集合 P は以下の生成規則からなる．

$$S \to AS', S \to S'B, S' \to \epsilon, S' \to 0S'1, A \to 0A, A \to 0, B \to B1, B \to 1$$

例えば， $x = 0^i 1^{i+j} (j \geq 1)$ は以下の様に生成される．

$$S \Rightarrow S'B \overset{*}{\Rightarrow} 0^i S' 1^i B \Rightarrow 0^i 1^i B \overset{*}{\Rightarrow} 0^i 1^i B 1^{j-1} \Rightarrow 0^i 1^i 1^j$$

0 の方が多い列も同様の考えで生成される．従って，列 x が問題の言語に入っていれば x は G_2 によって生成されるという方向の証明は良いであろう．

逆（つまり， G_2 によって生成されるなら，言語に属すような列しかあり得ない）の証明はちょっと面倒であるが，以下の事実から理解してもらえるであろう．最初に使える生成規則は $S \to AS'$ か $S \to S'B$ しかあり得ない．例えば前者を使ったとしよう．するとそのときの列は $S'B$ である．ここで，生成規則を良くみてもらうと， $B \overset{*}{\Rightarrow} x \in \Sigma^*$ であるなら， $x = 1^j (j \geq 1)$ しかあり得ないことが分かる．同様に $S' \overset{*}{\Rightarrow} y \in \Sigma^*$ なら $y = 0^i 1^i (i \geq 0)$ しかあり得ないことも分かる．従って， $S'B$ から導かれる終端記号のみからなる列は $0^i 1^i 1^j$ の形しかあり得ないことが結論される．最初に別の生成規則（$S \to S'B$）を使った場合も同様である．

上で， $A \overset{*}{\Rightarrow} x$ 且つ $B \overset{*}{\Rightarrow} y$ であるための必要十分条件が $AB \overset{*}{\Rightarrow} xy$ であることを使っている．これは一見当たり前の性質の様に見えるが，文脈自由文法の重要な性質の一つである．（解答終り）

練習問題 3.2　$G_4 = (\{S\}, \{0, 1\}, \{S \to \epsilon, S \to 0S0, S \to 1S1\}, S)$．どんな言語を生成するか．

解答　いくつか列を生成してみればすぐ分かる．例えば，

$$S \Rightarrow 0S0 \Rightarrow 01S10 \Rightarrow 010S010 \Rightarrow 0100S0010 \Rightarrow 01000010.$$

もう分かったであろう．すなわち $L(G_4) = \{xx^{\mathrm{R}} \mid x \in \{0, 1\}^*\}$ である．　（解答終り）

練習問題 3.3　*1* 章で最初に導入した「文章」は例えば

$$(5 + 7 \times 3)/(3 + 6)$$

のような数式であった． *1* から *9* までの数に関する正しい数式を生成する文脈自由文法を求めよ．

解答　上の例では $\times$ が $+$ より優先度が高いという普通の規則を使ってカッコを省略している．これを許すとちょっと難しいので，取りあえず，常にカッコを使用する場合を考えるのが良かろう．その場合，2 章で勉強した正規表現の定義が参考になるであろう．つまり，1 から 9 までの数字は「式」である．式 1 と式 2 が式ならば，(式 1+ 式 2) も式である．といった感じで式が定義できるのが分かるであろう．つまり，「式」を変数とするのである．これから文脈自由文法を導くのは困難ではない．（解答終り）

練習問題 3.4　$\{0^n \mid$ ある整数$k \geq 0$に対して$n = 2^k$と書ける$\}$ を生成する文法を求めよ．

解答　簡単である．$(\{S\}, \{0\}, \{S \to SS, S \to 0\}, S)$ で良いであろう．例えば，S が SS に置き換わって，それぞれの S が SS に置き換わるので $SSSS$ が導かれる．1 個の S は 1 個の 0 に置き換わるので，0^4 が得られる．

残念ながらこの解答は文法の定義を全く誤解している．この言語が生成する言語は単に 1 個以上の 0 からなる列全体である（例えば，000 の様な本来生成してはいけない列まで生成してしまう）．もう一回導出の定義を復習して欲しい．1 ステップの導出は 1 個の変数を置き換えれることであって，今の列に含まれる全ての変数を同時に置き換えなくてはならないとは一切言っていない．従って，上の文法の場合は，$S \Rightarrow SS \Rightarrow SSS \overset{*}{\Rightarrow} 000$ の様に 000 を導出してしまうのである．それでは正しい答えは何であろうか．実は存在しないのである．問題の言語は文脈自由文法では生成不可能である．（解答終り）

練習問題 3.5　今までよりちょっと難しい問題を考えてみよう．練習問題 *3.3* の言語の補集合（つまり $L = \{0,1\}^* - L(G_4)$）を生成する文法を求めよ．

解答　列 x が問題の言語 L にはいるための条件は以下の様に表せることを理解して欲しい．(i)x の長さが奇数，又は (ii) ある列 $\alpha, \beta \in \{0,1\}^*$ に対して，

$$x = \alpha 0 \beta 1 \alpha^{\mathrm{R}} \text{又は} x = \alpha 1 \beta 0 \alpha^{\mathrm{R}}$$

と書ける．条件 (ii) は列の両端から等距離のところに異なった記号が現れていることを意味している（長さが偶数で yy^{R} の形で無いなら必ずこのような一組の記号が存在するはずである）．この様に考えると以下の文法にたどり着くことは困難ではない．なお，今後は文法を与えるときは生成規則の集合のみを与えることで行う．特に断わらない限り開始記号は常に S を使う．また，読者は既に気づいておられることと思うが，慣習として，変数には英語の大文字を使い，終端記号には英語の小文字や 0 や 1 を良く使用する．

$$\begin{aligned}
&S \to S_1, S \to S_2,\\
&S_1 \to 0, S_1 \to 1, S_1 \to 00S_1, S_1 \to 01S_1, S_1 \to 10S_1, S_1 \to 11S_1,\\
&S_2 \to 0S_20, S_2 \to 1S_21, S_2 \to 0A1, S_2 \to 1A0, A \to \epsilon, A \to 0A, A \to 1A,
\end{aligned}$$

S から始まって，最初の生成規則として，$S \to S_1, S \to S_2$ のどちらを使うかで上の条件のどちらを満たす列を生成されるかが決まる．$S \to S_1$ を使うと S_1 からは長さが奇数の列が（どんな列でも）生成されることが分かるであろう．S_2 からは上の条件の (ii) の方の列が生成される．そのためには以下の導出過程が分かれば十分である．

$$s_2 \overset{*}{\Rightarrow} xS_2x^{\mathrm{R}} \begin{cases} \Rightarrow x0A1x^{\mathrm{R}} \\ \Rightarrow x1A0x^{\mathrm{R}} \end{cases}$$

なお，A からは任意の列が導出される．（解答終り）

ところでこの練習問題 3.5 は文脈自由文法に関する重要な性質を紹介している．すなわち，上の言語は条件 (i) を満たす言語と (ii) を満たす言語の和集合という形で与えられるのであるが，上の生成規則の最初の 2 つの規則 $S \to S_1, S \to S_2$ が正にその和集合を作るために使われているのである．従って，次の定理には証明が必要ないであろう．

定理 3.1　言語 L_1 と L_2 が文脈自由文法で生成されるならそれらの和集合もまた文脈自由文法で生成される．

次の性質も非常に重要である．以前，導出の定義を述べるときに，2 つ以上の変数が存在したら，いずれの変数に対して生成規則を適用しても良いと言った．しかし，次の定理によれば，そのような時に最も左の変数に生成規則を適用するという制限を付加しても一般性を失わないことが分かる．このように，一番左の変数を常に置き換えるような導出を**最左導出**と呼ぶ．

定理 3.2　文法 G によって，終端記号のみからなる列 x が導出されるとき，x は最左導出によっても導出される．

証明　以下の様な導出を考える．

$$S \overset{*}{\Rightarrow} \cdots A \cdots B \cdots \overset{*}{\Rightarrow} \cdots A \cdots \beta \cdots \overset{*}{\Rightarrow} \cdots \alpha \cdots \beta \cdots \overset{*}{\Rightarrow} x_1 \alpha x_2 \beta x_3$$

途中の列に変数 A と B が現れた時に，ここでは B を先に β で置き換えて，後から A を α で置き換えている．しかし，他の部分の置き換えを全て同じものを使うと仮定するなら，この A と B の置き換えを逆にしても全く同じ列 $x_1 \alpha x_2 \beta x_3$ が導出されるということが容易に分かるであろう．つまり，A の左の部分が最終的に x_1 になって，A と B の間の部分が最終的に x_2 に，B に右の部分が最終的に x_3 になる．これらの部分は，A と B の置き換えの順番には依存しないからである．この説明は単に 2 個の変数の置き換えの順番であったが，この議論の単純な拡張によって定理が証明できる．（証明終り）

練習問題 3.6　以下の文法 G_6 はどんな言語を生成するか．

$$S \to SS, S \to \epsilon, S \to aB, S \to bA, A \to a, A \to bAA, B \to b, B \to aBB$$

解答　$L(G_5) = \{x \mid x \in \{a,b\}^* \text{且つ} \sharp_a(x) = \sharp_b(x)\}$ である．例えば，列 $aaabaabbbb$ がどのように導出されるか見て見よう．

$$\begin{aligned} S &\Rightarrow aB \Rightarrow aaBB \Rightarrow aaaBBB \Rightarrow aaabBB \\ &\Rightarrow aaabaBBB \Rightarrow aaababBB \Rightarrow aaababbB \Rightarrow aaababbb \end{aligned}$$

この導出は最左導出になっていることが分かるであろう．ここで最左導出を使ったのは大きな意味がある．というのは，この導出は目標の列 x（今の場合は $aaabaabbbb$）が与えられると，導出の過程が以下の意味で一意に決まるからである．最初の置き換えは $S \to aB$ を使っているが，その理由は x の最初の記号が a だからである．（$S \to bA$ を使ってしまったら，生成される列は必ず b から始まってしまうことが確定されてしまうので，今の目標の列 x は導出できない．）次の置き換えに $B \to aBB$ を使っている理由も分かるであろう．つまり，置き換えようとする（一番左の）変数が B で，その場所に対応する（つまり左から 2 番目の）x の記号が a であるからこの置き換えしか考えられないのである．

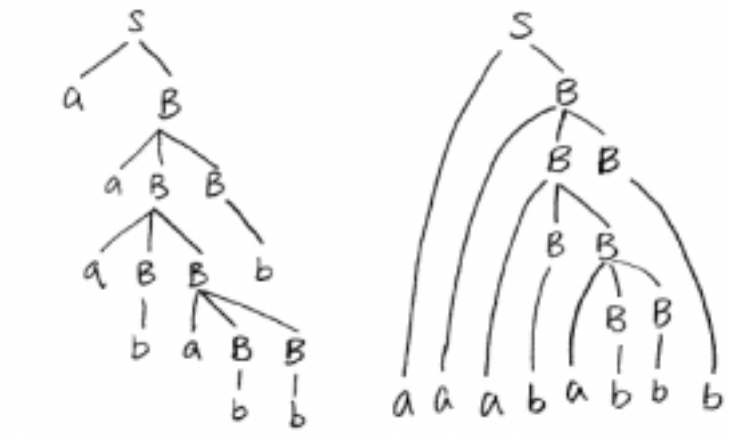

図 3.0 導出木

更に各生成記号の意味を理解してほしい．ここで，変数 A（B）はその変数を「将来 a（b）で置き換えて欲しい，あるいはそのように置き換えればバランスする」ということを意味している．従って，例えば $S \to aB$ は，最初に（a も b も 0 個でバランスしているところから）1 個の a を確定させるにはバランスを保持するために B を 1 個付けておく必要がある． $B \to aBB$ は，本来 b で置き換えたい変数なのに，逆の a を確定させてしまったので，2 個の B を後に付けておく必要があるわけである．以上の説明はアイデアのみで，とても証明とは言えないが，答えの正しさが分かっていただけると思う．なお，生成規則の最初の $S \to SS$ はなぜ必要なのか考えて欲しい． （解答終り）

この節の最後に便利な道具である**導出木**を紹介する．導出木は文脈自由文法とある導出が与えられると決まるものである．例えば，上に問題の解答で現れた列 $aaababbb$ の導出に対しては，図 3.0 の左側の様な導出木が対応する．導出木は，ある変数（今の場合は S）を根として，葉を左から右に読むと導出された列（今の場合は $aaababbb$）になっている様な木である．木の定義を忘れてしまった人は本書の最初の 1.1 を復習して欲しい．ここで，葉を「左から右に読む」というのが分かりずらいかも知れないが，その場合は同図の右側の様に，葉を下の方に延ばして考えると分かりやすいであろう．

この導出木と上の導出の関係については明らかであろう．例えば，最初に $S \to aB$ が使われて，木では S の下が 2 つに分岐している．次に $B \to aBB$ が使われて B が 3 つに分岐している．ここで， B が 2 個になって，次にどちらかの B が置き換わるわけであるが，導出木に場合は，どちらを先に置き換えたかという情報が存在しない．言葉を買えれば，どちらを先に置き換えても結果は変わらないということである，従って，最左導出を常に用いても良いのは明らかで，上の定理の別証明になっている．

3.3 文脈自由文法と正規言語

文脈自由文法を導入した理由の一つが，正規表現や有限オートマトンより能力の高いシステムが欲しかったことであることを思い出して欲しい．

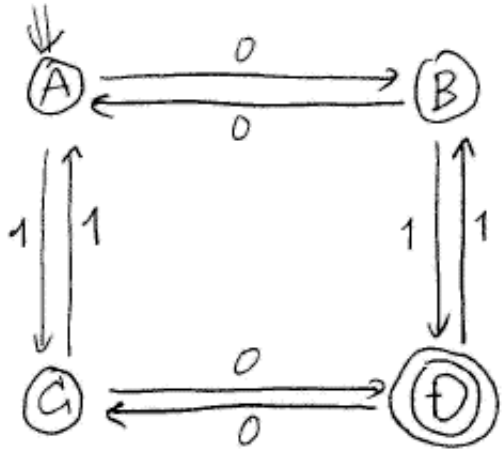

図 3.1　L を認識する有限オートマトン

前節で文脈自由文法は $\{0^n1^n \mid n \geq 0\}$ の様な正規言語でない言語を生成できることが分かった．しかし，全ての正規言語を生成できるかどうかはまだ述べていない．文脈自由文法が正規表現や有限オートマトンよりも真に強力であることを示すためにはこのことを言わなければいけない．

定理 3.3　L が正規言語であるなら，L を生成する文脈自由文法が存在する．

証明　簡単なので例を使って証明する．L は正規言語であるからそれを認識する fa が存在する．その fa が図 3.1 の様なものであったとしよう．この fa は前節の図 2.2 の例と類似していて，0 の個数と 1 の個数が共に奇数であるような列を受理する．この fa に対して我々は以下の様な生成規則を用意する．

$$
\begin{aligned}
&A \to 0B,\ A \to 1C,\ B \to 0A,\ B \to 1D \\
&C \to 0D,\ C \to 1A,\ D \to 0C,\ D \to 1B \\
&B \to 1,\ C \to 0
\end{aligned}
$$

生成規則の作り方はほぼ説明を要しないであろう．変数は fa の状態に対応している．また，生成規則については，例えば最初の $A \to 0B$ は状態 A から B へ記号 0 による遷移が存在することに対応している．また，最後の $C \to 0$ は，状態 C から記号 0 で受理状態に行くことに対応している．初期記号としては，fa の初期状態に対応する A を選択する．

この構成の正しさ（つまりこの文法が fa と同じ言語を生成すること）もほぼ明らかである．例えば，文法の方で以下の導出を考えてみよう．

$$A \Rightarrow 0B \Rightarrow 00A \Rightarrow 001C \Rightarrow 0010$$

これは fa では初期状態 A からスタートして，0 を読んで B に行き，また 0 を読んで A に戻り，1 を読んで C に行き，最後に 0 を読んで受理状態に到達する動作に対応している．（証明終り）

3.4 文脈自由文法の標準形

文脈自由文法の生成規則に関しては，左辺が 1 個の変数であること以外には制限は無い．しかし，右辺にも制限を加えた幾つかの標準形が知られている．この節ではそのような標準形の幾つかを紹介する．

チョムスキーの標準形とグライバッハの標準形を紹介する．前者は右辺がただ 1 個の終端記号か，2 個の変数のみの生成規則のみにできるというもので，見かけは非常によい．ただ，この場合，見かけの良さ以外はあまり重要な応用は（筆者の知っている範囲では）見当たらない．それに対して，グライバッハの標準形は右辺が左端の 1 個の終端記号と 0 個以上の変数からなるというもので，一見重要性がわからないかも知れない．しかし，以下で述べるように幾つかの重要な応用がある．

例えば以下の生成規則を考えてみよう．（変数には大文字のローマ字を使い，終端記号には数字やローマ小文字を使うという我々の慣習を思い出して欲しい．）

$$S \to AX,\ S \to CC,\ X \to SB,\ A \to 0,\ B \to 1,\ C \to 2$$

これは左辺が 1 個の終端記号か又は 2 個の変数なのでチョムスキーの標準形になっている．それに対して以下の生成規則は，右辺が必ず終端記号から始まっているのでグライバッハの標準形である．

$$S \to 0SB,\ S \to 2A,\ A \to 2,\ B \to 1$$

実はこれらの文法は同じ言語を生成する．そこで，本書の最初で述べた文法の目的に沿って，例えば，$x = 00221$ という列がこれらの文法によって導出されるかどうかを調べたいとしよう．1 番目の文法であったなら，どのように調べたらよいかすぐにはわからないであろう（例えば，最初に適用すべき規則が上記の規則集合のの一番左なのか，2 番目なのかがそれらを見ただけではすぐには分からない）．しかし，2 番目の文法なら，以下のようにしてすぐに分かるのである．（練習問題 3.6 でも似たような議論を行った．）調べたい列 x の最初の記号が 0 であるから，最初に適用すべき規則も右辺が 0 で始まる $S \to 0SB$ しかありえない．2 番目に適用すべき規則に関しても同様である．このようにして適用する規則を決めていくなら，結局以下のような導出しかありえないことになる．

$$S \Rightarrow 0SB \Rightarrow 00SBB \Rightarrow 002ABB \Rightarrow 0022BB \Rightarrow 00221B$$

しかし，この導出の最後の列にはまだ変数が残っているので，x は導出できないと結論してよい．つまり，右辺の最初が終端記号で始まっていることは，目標とする列が分かっているときに，適用すべき生成規則の選択において大きなヒントになるのである．

グライバッハの標準形は各生成規則が正確に 1 個の終端記号を含んでいるので，長さ n の列を導出するなら規則の適用回数も正確に n 回である．このこともグライバッハの標準形の大きな利点で次の章の議論で非常に重要になってくる．さて，これら 2 つの標準形に関する定理を紹介するが，ここで重要な仮定を設けることにする．それは，この節で扱う文脈自由文法は（標準形であるとないとにかかわらず）$A \to \epsilon$ の形の（つまり右辺が ϵ の）生成規則を含まないとする（章末の問題参照）．従って，当然 ϵ を導出することも不可能であるから，この章での文脈自由言語には ϵ は含まれない．言語に ϵ が入っているかどうかは，少なくとも現実的にはそれほど重要では無いので，この制限が大きく一般性を失うことは無い．

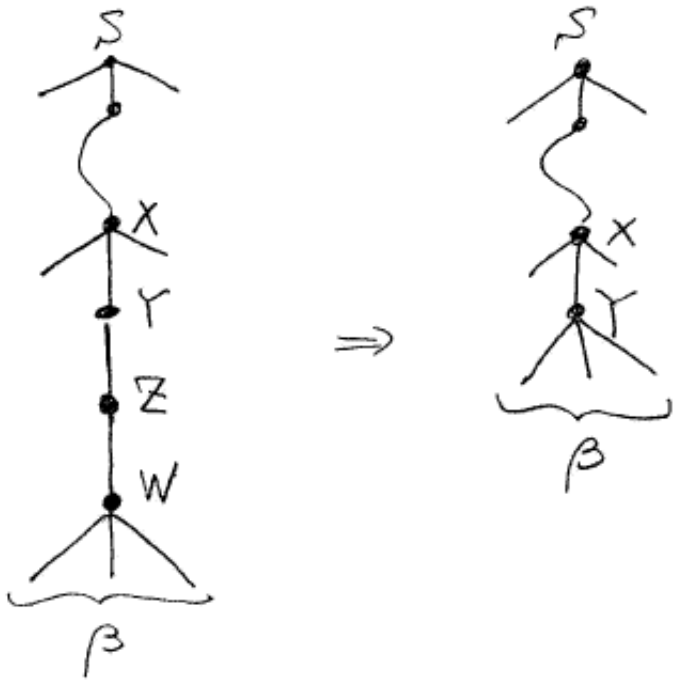

図 3.2 単一規則の除去

定理 3.4　L が文脈自由言語であるなら，L を生成するチョムスキーの標準形文法が存在する．

証明　L を生成する文法が以下のような生成規則を含んでいると仮定しよう．

$$A \to B,\ B \to C,\ C \to A,\ A \to aABa,\ \cdots$$

これらの規則はいずれもチョムスキーの標準形の条件を満たしていない．我々はまず，右辺が 1 個の変数の形の規則（単一規則と呼ぶ）を除くことにしよう．そのためにはまず，任意の 2 つの異なる変数 X, Y に関して，$X \overset{*}{\Rightarrow} Y$ であるかどうかを調べる．上の例の場合であれば，単一規則そのものである $A \overset{*}{\Rightarrow} B$，$B \overset{*}{\Rightarrow} C$，$C \overset{*}{\Rightarrow} A$ 以外にも，単一規則を 2 ステップ以上使って $B \overset{*}{\Rightarrow} A$ 等も可能であることに注意されたい．単一規則の個数を N としたとき，N ステップ程度までを調べれば十分であることが分かる（この部分は次の練習問題でもう少し詳しく説明する）．$X \overset{*}{\Rightarrow} Y$ であることが分かったら，左辺が Y の非単一規則 $Y \to \alpha$ 全てに対して，新たな規則 $X \to \alpha$ を追加する．例えば，上の例なら，例えば，$B \overset{*}{\Rightarrow} A$ であるから，$B \to aABa$ を追加する．この作業の後で全ての単一規則を除く．

こうして単一規則を排除することができたが，そのことによって生成される言語が変化しないことを知るには導出木をイメージしてもらえば分かりやすい．図 3.2 の左側のように Y から単一規則が連続していても（$Y \to Z,\ Z \to W$）いずれは必ず非単一規則（$W \to \beta$）が使われるわけだから，同図右のように Y にその非単一規則を直接適用（$Y \to \beta$）してしまえばよい．また上の変形によってそれが可能になっている．

この段階で右辺の長さが 1 の規則の右辺は終端記号になっているからそれは，標準形の条件を

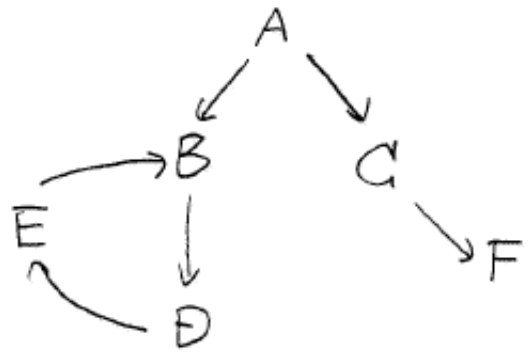

図 3.3 単一規則のグラフ的表現

満たしている．そこで，$A \to aABa$ の様な規則を処理しよう．これは簡単で，まず各終端記号 t に対して新たな変数 X_t を導入して，$X_t \to t$ の形の規則を追加し，その後で，各規則の右辺の終端記号 t を変数 X_t で置き換える．上の例なら $A \to aABa$ を $A \to X_aABX_a$ に変形する（更に $X_a \to a$ の規則が追加される）．こうして，長さ 2 以上の右辺は全て変数だけになった．長さ 3 以上の右辺を持つ規則，例えば，$A \to BCDE$ は更に新たな変数 Y_1, Y_2 を導入して，$A \to BY_1$, $Y_1 \to CY_2$, $Y_2 \to DE$ の 3 つの規則で置き換える．この作業の正当性はほぼ明らかであろう．（証明終わり）

練習問題 3.7　以下の生成規則で与えられる文法をチョムスキーの標準形に直せ．初期記号は A とする．

$$A \to B,\ A \to C,\ B \to D,\ D \to E,\ E \to B,\ C \to F,\ B \to b,\ E \to ADa,\ C \to ABB$$

解答　単一規則だけを使って A から到達できる変数はどれかを調べるためには，与えられた単一規則の集合を図 3.3 の様に図示してみると分かりやすい．その上で，例えば変数 B から単一規則のみを使って到達できる変数を調べるためには以下のようにすれば良い．(1) まず B に印を付ける．(2) 今印のついている変数から矢印が向かっている変数（例えば B からなら D，2 つ以上ある場合もあるので注意）にまだ印が付いていないなら印を付ける．(3)(2) の作業をこれ以上新しい変数に印が付かなくなるまで繰り返す．

今の例の場合，変数 A からは全ての変数に到達できることが分かる．従って，上の作業規則に従って，

$$A \to b,\ A \to ADa,\ A \to ABB$$

という規則を追加される規則の集合になる．同様に B からは B，D，E に到達可能であるから，

$$B \to b,\ B \to ADa$$

が追加される規則になる．こうして全ての変数に対して追加する規則を得る．その後単一規則を除く．（この例の場合は変数 F はそれを左辺に持つ非単一規則が存在しない．こういう変数は

始めから不要であることが分かる．）単一規則さえ除いてしまえば後は簡単で結局最終的な規則集合は以下のようになる．（各自確かめること）

$$
\begin{aligned}
&A \to b,\ A \to AY_1,\ Y_1 \to DX_a,\ A \to AY_2,\ Y_2 \to BB \\
&B \to b,\ B \to AY_3,\ Y_3 \to DX_a,\ D \to b,\ D \to AY_4,\ Y_4 \to DX_a \\
&E \to b,\ E \to AY_5,\ Y_5 \to DX_a,\ C \to AY_6,\ Y_6 \to BB \\
&X_a \to a,\ X_b \to b
\end{aligned}
$$

（解答終り）

次がグライバッハの標準形であるが，念のためより形式的な定義を与えておこう．全ての生成規則が $A \to a\gamma$，但し a は終端記号で γ は変数だけからなる長さ 0 以上の列，の形に書けるときその文法はグライバッハの標準形と呼ばれる．

定理 3.5　L が文脈自由言語であるなら，L を生成するグライバッハの標準形文法が存在する．

証明　きっちり証明して且つその本質を十分理解しようとするとかなりやっかいである．（実はこの定理はできれば避けて通りたかった．しかし，次章のプッシュダウンオートマトンとの関わりでどうしても述べておかねばならないのである．）そこで以下では証明のアイデアを直観的に述べるにとどめることにする．

前定理から，L はチョムスキーの標準形によって与えられていると仮定して一般性を失わない．そこで，この節の始めに現れた以下の規則を復習して見よう．

$$S \to AX,\ S \to CC,\ X \to SB,\ A \to 0,\ B \to 1,\ C \to 2$$

ここで後半の 3 つの規則は既にグライバッハの標準形の条件を満たしている．前半 3 つは満たしていない．例えば 3 番目の $X \to SB$ の右辺は変数で始まっているので，これを終端記号で始まる様に直さなければいけない．ここでも導出木をイメージすると分かりやすい．図 3.4 に示すように，$X \to SB$ が使われた後で次に置き換えられるのは（最左導出なら）変数 S である．その置き換え方は $S \to AX$ によるか $S \to CC$ による 2 通りある．前者の場合は次に置き換えられる変数は A で，A の置き換えは終端記号 0 によるものだけである．後者の場合も同様で，この図の右側に示してある．ここで左側の導出木を良くみると，変数 X を置き換えるところで，一気に $0XB$ で置き換えてもよいことが分かる．つまり，$X \to 0XB$ という規則を新たに導入するのである．同様に右側の導出木の場合は $X \to 2CB$ という規則を導入する．これら 2 個の規則を導入すると，元の規則 $X \to SB$ はもはや必要ないことが分かる．新たに導入された規則は標準形の条件を満たしているので，満たしていない規則の数を 1 個減らすことができた．同様の作業を残り 2 つの条件を満たしていない規則に対して実行すれば条件を満たさない規則は無くなってしまう．

要するに右辺が変数で始まっている規則 P に対して，上のような導出木をつくるのであるが，一番左の変数のみを他の規則を使ってどんどん置き換えていく．変数は最終的には終端記号に置き換わるはずであるから，そうなるまで可能な全てを尽くす．一番左のパスが終端記号で終ったら，その導出木全体を 1 個の規則と見なして，その規則を追加し，元の規則 P を取り去ってしまう．簡単であると思われるかもしれない．

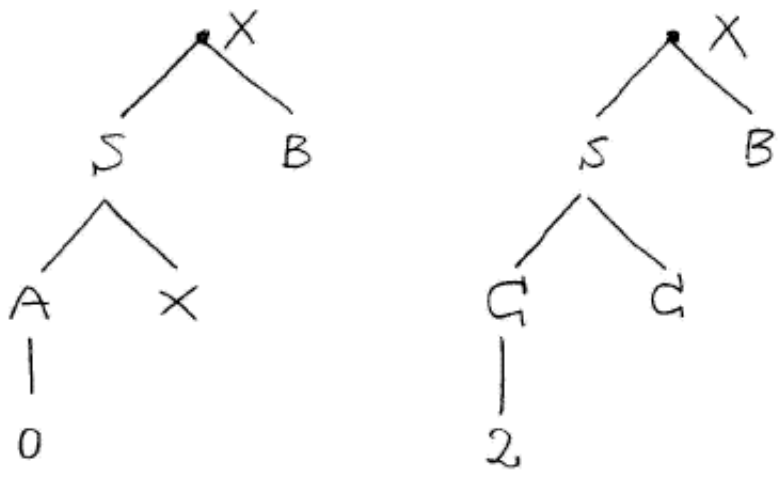

図 3.4 規則の右辺の最初の変数の置き換え

このアイデアは大筋で正しいのであるが，一点大きな困難がある．以下の規則集合を考えてみよう．

$$A \to CB,\ B \to AC,\ B \to AA,\ C \to BA,\ A \to a,\ C \to b$$

上記と同様に，変数 A から始めて導出木の左端の枝をどんどんたどって行ってみよう．すると，

$$A \Rightarrow CB \Rightarrow BAB \Rightarrow AAAB \Rightarrow$$

のように変数 A が再び先頭に現れてしまうのである．つまり， A から始めて最左導出を続けていくと，いつの段階で一番左の変数を終端記号に置き換えたら良いかが分からないのである．（以前の例では，最左導出をしていくと，ある時点で終端記号に置き換えるしか無かった．）このような構造は図 3.5 を見るとより分かりやすい．この図では，各変数を頂点と見立て，その変数がどのように置き換えられるかを図示している．上の置き換えの例も， A からスタートして，上の導出の各ステップにおける左端の変数である C， B， A とたどって行くことに対応していることが分かっていただけるであろう．この時，たどった枝の記号の列が右から左に並んでいることにも気が付いて欲しい．何かに似ていないであろうか．そう，前節でやった有限オートマトンの文脈自由文法による模倣とほとんど同じである．（つまり，例えば導出 $A \overset{*}{\Rightarrow} BAB$ は状態 A から列 BA を読んで状態 B に至ると考える．）

一般的に言って，ある変数 Z からスタートして，チョムスキー標準形規則の中の右辺が 2 個の変数であるもの（これを膨張規則と呼ぼう）のみを使用して最左導出を行って $Z \overset{*}{\Rightarrow} \gamma$ を得たとしよう．すると上の議論から分かるように，可能な γ の全体（これを $V(Z)$ と書くことにしよう）は変数集合上の正規言語になる．正規言語であるから当然それを表現する正規表現が存在する．今の例の場合であるなら，変数 A からスタートしたときの $V(A)$ は以下の正規表現で表される．

$$A(AAB + CAB)^* + CB(AAB + CAB)^* + BAB(AAB + CAB)^*$$

ここで， 3 つの項のうち，最初の項は図 3.5 のオートマトンの状態（変数に対応） A で終る列

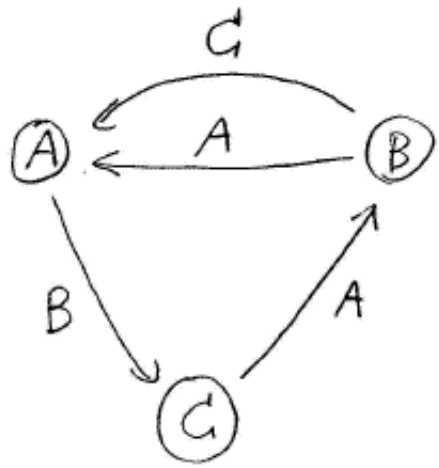

図 3.5 規則の有限オートマトンによる表現

の集合，第 2 の項は状態 C で終る列の集合，第 3 の項は状態 B で終る列の集合を表していることが分かるであろう．

変数 A からスタートして膨張規則によって最左導出を繰り返していくと，いつかは膨張規則をやめて終端記号で置き換えないといけない．つまり，

$$A \overset{*}{\Rightarrow} X_1X_2\cdots X_n \Rightarrow qX_2\cdots X_n,\ \ X_i \in \{A,B,C\},\ q \in \{a,b\}.$$

である．この時，上の議論から，

$$X_1X_2\cdots X_n \in V(A)$$

であり， $A \to a,\ C \to b$ という規則があるので， $X_1 = A$ なら $q = a$，また $X_1 = C$ なら $q = b$ でなければならないことが分かる（$X_1 = B$ ということはあり得ない）．ということは，もし，規則の右辺に正規表現を書くことが許されているなら，

$$A \to a(AAB + CAB)^*,\ A \to bB(AAB + CAB)^*$$

のような規則を導入すれば右辺の最初を終端記号にできるであろう．つまり，正規表現 R に対して， $A \to R$ という規則があるなら，任意の列 $\alpha \in L(R)$ に対して $A \Rightarrow \alpha$ が 1 ステップで導出できると考えるのである．

勿論右辺に正規表現をかくことは許されていない．そこで，それと同等のことを以下の様に行う．つまり，

$$A \to a(AAB + CAB)^*$$

の代りに

$$A \to a,\ A \to aZ_1,\ Z_1 \to AAB,\ Z_1 \to CAB,\ Z_1 \to AABZ_1,\ Z_1 \to CABZ_1$$

を使うのである．つまり，正規表現のスター演算に対して 1 個の新しい変数を導入して，スターの繰り返し構造をうまくシミュレートしてしまうのである．例えば，正規表現 $a(AAB+CAB)^*$

は列 $aAABCABCAB$ を含む．これは，

$$A \Rightarrow aZ_1 \Rightarrow aAABZ_1 \Rightarrow aAABCABZ_1 \Rightarrow aAABCABCAB$$

の様に新しい生成規則によって導出される．ここで，$Z_1 \to AABZ_1$ という規則は $Z_1 \to Z_1AAB$ と書いても同じ効果であるが，絶対に絶対にそう書いてはいけない．理由はすぐに説明する．同様に

$$A \to bB(AAB + CAB)^*$$

の代りに

$$A \to bB,\ A \to bBZ_2,\ Z_2 \to AAB,\ Z_2 \to CAB,\ Z_2 \to AABZ_2,\ Z_2 \to CABZ_2$$

を導入する．

以上で A が左辺に来る規則の処理が終った．同様の作業を B が左辺に来る規則，C が左辺に来る規則に対しても実行する．つまり，正規表現を求めて，スターの部分に対して新しい変数を導入して，それらに関する規則を上と全く同様に求める．

$$B \to aA, B \to aC, B \to aAZ_3, B \to aCZ_3, Z_3 \to ABC, Z_3 \to ABA, Z_3 \to ABCZ_3, Z_3 \to ABAZ_3$$

$$B \to bBA, B \to bBC, B \to bBAZ_4, B \to bBCZ_4, Z_4 \to ABC, Z_4 \to ABA, Z_3 \to ABCZ_4, Z_3 \to ABAZ_4$$

$$C \to b,\ C \to bZ_5,\ Z_5 \to BAA,\ Z_5 \to BCA,\ Z_5 \to BAAZ_5,\ Z_5 \to BCAZ_5$$

$$C \to aAA, C \to aCA, C \to aAAZ_6, C \to aCAZ_6, Z_6 \to BAA, Z_6 \to BCA, Z_6 \to BAAZ_6, Z_6 \to BCAZ_6$$

これらを導入したあとは，元の 4 個の膨張規則を全て取り去ってしまっても生成する言語に変化は生じないことが分かる（各自証明を試みよ）．

まだ終ったわけではない．新たに導入した変数（Z_1, Z_2 等）を左辺にもつ規則の右辺が変数で始まっているからである．しかし，これらの左端の変数は新たに導入された変数ではなく，元からある変数なので，最初に述べた，どんどん置き換えるという作業をすればいずれは必ず終端記号が最初に来る様になる．実は，この部分はそれほど単純ではない．というのは上で導入した正規表現は一般にはこんな簡単なものではなく，スター演算が入り組んだ構造になっている可能性があるからである．しかし，それでも内部のスターから外に向かって順番に新しい変数を導入してスターをはずして行けば良い（章末の問題参照）．この場合，新しい規則の右辺の最初の変数が新しく導入した変数になる可能性があるが，その変数はより内部のスターに対する変数である．つまり，問題になったような変数が元に戻ってくるという可能性が無いのである．（ここで，上で言った $Z_1 \to Z_1AAB$ としてはいけない理由が分かったであろう．）例えば，

$$Z_1 \to CAB$$

の代りに

$$Z_1 \to bZ_5AB,\ Z_1 \to aAAZ_6AB,\ Z_1 \to aCAZ_6AB$$

を置けば良い．（証明終り）

3.5 文脈自由文法で生成できない言語

文脈自由文法は有限オートマトンや正規表現より真に能力が高いことが分かったが，これであらゆる言語を表現するのに十分なのであろうか．実は，まだ十分ではないのである．

この節では文脈自由文法では生成できない言語が存在することを示す．有限オートマトンの場合は記憶が有限という性質を使って比較的簡単に認識できない言語の存在をしめすことができた．文脈自由文法の場合はそれほど簡単ではない．ヒントとなるのは前章の章末問題で考えた正規表現の能力の限界を示す性質である．そこで示された様に，正規表現では列の繰り返しを表現する能力に大きな制限があり，直観的にはある部分の繰り返しと他の部分の繰り返しの間に関連をつける（例えば繰り返し回数を同じにするとかの）ことができない．文脈自由文法の場合はこのような複数の繰り返し構造に関連を付けることはできるのであるが，その関連づけの能力に大きな制限がある．その制限を端的に示す次の定理を紹介しよう．

定理 3.6 L を有限でない文脈自由言語とする．すると，L によって決まるある定数 K が存在して，L に属す長さ K 以上の任意の列 z に対し，以下の 3 条件を満たす 5 個の列 u, v, w, x, y が存在する．*(i)* $z = uvwxy$ と書ける．*(ii)* $vx \neq \epsilon$．*(iii)* 任意の $i \geq 1$ に対して $uv^iwx^iy \in L$.

この定理を始めて見たとき何を言っているのかよく分からないという読者がおられると思う．そこで，証明の前に，この定理の応用例を先に見てもらうことにしよう．そうすれば，定理が何を言っているのかがよりはっきりするに違いない．なお，この定理はその形から uvwxy 定理と呼ばれている．

練習問題 3.8 $L = \{a^nb^nc^n \mid n \geq 1\}$ は文脈自由言語ではないことを示せ.

解答 L が文脈自由言語であると仮定しよう．すると，uvwxy 定理が成立する．まず，L によって決まる定数 K であるが，この値自体は分からない．しかし，L に属す列 $a^nb^nc^n$ は n の値を大きくすることによっていくらでも長くできるので，やがては必ず K を超える長さのもの（例えば $a^Kb^Kc^K$）が必ず取れる．定理によれば，この列 $a^Kb^Kc^K$ は必ず上の条件の (ii) と (iii) を満たす形で 5 個の部分列 u, v, w, x, y に分解できる．条件 (ii) は x と v の双方が共に ϵ になることは無いと言っているので，v $\neq \epsilon$ と仮定してみよう（x $\neq \epsilon$ の場合も同様の証明ができるので省略する）．すると以下に示すように，w をどのように分解したとしても条件 (iii) は絶対に満たすことができないのである．つまり，定理で保証している分解ができないのであるから矛盾である．

さて，$a^Kb^Kc^K$ の分解の仕方であるが，ϵ でない v をどの部分にとるかによって場合わけする．

（場合 1）図 3.6 (i) の様に v を a と b の境界にとる．つまり，$v = a^ib^j$, $i \neq 0$, $b \neq 0$ となるようにとる．この場合他の u, w, x, y をどのように取ろうとも uv^2wx^2y は $a \cdots b \cdots a \cdots b \cdots\cdots$ のような列になってしまい明らかに L には入らないのでこのような分解は不可能である．v を b と c の境界に取った場合も同様である．

（場合 2）図 3.6 (ii) の様に v を a の内部に取る．つまり $v = a^i$, $i \neq 0$ となるようにする．つぎに x をどこに取るかを考える．場合 1 と同様の理由で a と b あるいは b と c の境界に取ること

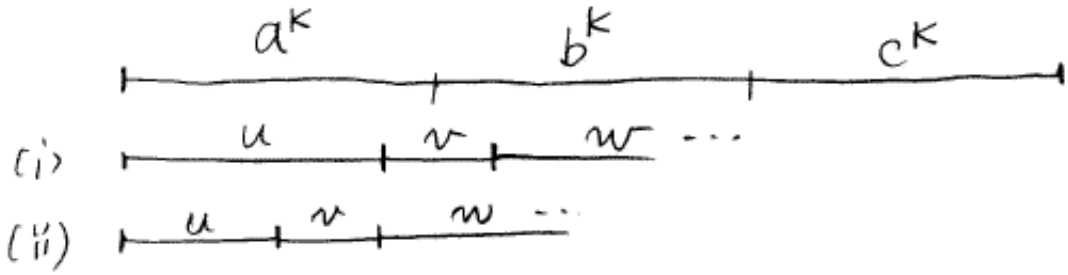

図 3.6 列の分割

はできないので，a または b または c の内部に取ることになる．例えば b の内部に取ったとしてみよう．すると uv^2wx^2y では a の個数と b の個数は増えるが c の個数は増えない．従って，この列は明らかに L には入らない．x を他の場所に取った場合も同様に記号の個数にアンバランスが生じてしまうことは容易に分かるであろう．

（場合 3）v を b または c の内部に取る．場合 2 とほとんど同様に uv^2wx^2y が L に入りえないことが示せる．　（解答終わり）

定理 3.6 の証明　以下の例でアイデアを示そう．

$$S \to aXA,\ S \to bXB,\ X \to aYA,\ X \to bYB,\ Y \to aXA,\ Y \to bXB$$

$$A \to a,\ B \to b,\ X \to aa,\ X \to bb,\ Y \to aa,\ Y \to bb$$

例えば，列 $aabaabaa$ を導出する導出木は図 3.7 (a) の様になる．この導出木の根から葉までのパスの中で最も長いものを見ると，変数 X が 2 回現れている．ここで，2 回目の X から下の部分木で導出される列（$= aa$）を w と置く．更に 1 回目の X から下の部分木で導出される列で上の w の左側の列を v，右側の列を x，更に根の S から導出される全体の列で上の v の左側の列を u，x の右側の列を y と置く（図参照）．こうして，列 $aabaabaa$ を $uvwxy$ の 5 個の部分列に分割することができた．ここで，以下の 2 点に注意されたい．

(i) 導出木の最長パスの長さを n としよう．すると，上の例の文法では，生成規則の右辺の長さの最長が 3 であるから，導出できる列の長さは 3^n で押えられることが分かる（つまり，各頂点で 3 分岐する高さ n の木の葉の数の上限である）．そこで，導出すべき終端記号の列として長さが 3^6（この値が定理で言っている K に対応する）以上のものを選べば，最長パスの長さは 6 以上になる．（いま考えている文脈自由言語は無限集合であるからこの様な長さの列は確かに存在する．）最長パスの長さが 6 以上なら，いま考えている文法の変数の数は 5 個であるから，鳩ノ巣原理によって，必ず同一の変数がそのパス上に 2 回現れる．つまり，上で述べたような状況が必ず生じるのである．

(ii) 定理 3.4 の議論によって規則には右辺がただ 1 個の変数であるような単一規則が存在しないと仮定してよい．つまり，上の様に v と w を選んだとき，少なくとも一方が ϵ でないと仮定してよい．

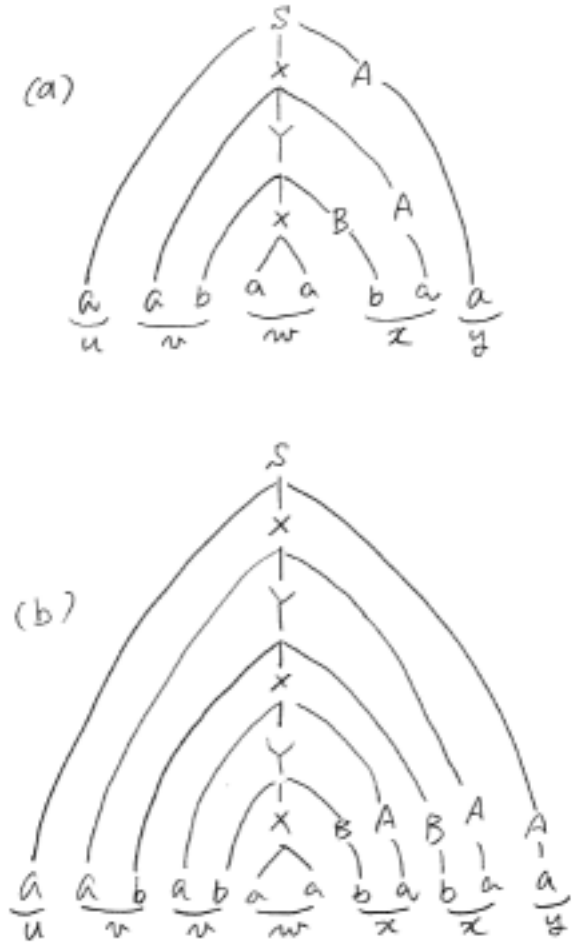

図 3.7 uvwxy 定理の証明

さてここで図 3.7 に戻ろう．2 回目の X から下の導出を今のものから変更して，1 回目の X から下の導出に置き換える．すると全体の導出木は同図 (b) の様になって，導出される列は明らかに uv^2wx^2y になる．この議論を繰り返せばいくらでも大きい整数 i に対して uv^iwx^iy の形の列を導出することができる．つまり，定理が証明されたことになる．（証明終り）

練習問題 3.9 $L = \{zz \mid z \in \{a, b\}^*$ が文脈自由言語で無いことを示せ.

解答 まずこの言語の特色をつかんで欲しい．この言語に入る列はその前半と後半が全く等しい（例えば $abaaabaa$）列である．これに対して，後半が前半の逆になっている列は練習問題 3.3 で見たように文脈自由言語である．2 つの言語は似ている様にも見えるが，文脈自由文法で生成できるかどうかと言う点では大きく異なっている．

さて，前と同じように uvwxy 定理を使おう．列 $a^Kb^Ka^Kb^K$ は明らかに L に入っていて，しかもその長さは K より大きい．従って，定理で言われている様に分割できるはずである．しかし，

$$u = a^K,\ v = b^k,\ w = \epsilon,\ x = a^K,\ y = b^K$$

と置くと

$$uv^2wx^2y = a^K b^{2k} a^{2K} b^K$$

となってしまい，L に入っていない．よって定理が成立することに矛盾する．

残念ながらこの証明は間違っている． uvwxy 定理の言っていることを復習してみよう．定理は「L が文脈自由言語なら，L に入る十分長いどんな列 z も条件を満たす形で分割できる」と言っている．従って，対偶をとれば，「条件を満たす形で分割できない十分長い列が存在すれば，L は文脈自由言語ではない」ということになる．上の主張は，十分に長い L に属すある列を取っているので，この点では問題ない．これが，分割できないことを言えば良いのであるが，ここのところが全く不十分であることが分かるであろう．つまり，分割できないことを言うためにはどんな分割をしても所定の条件を満たさないことを言わなければいけない．上の主張は，ある一つの分割をしたら条件を満たさないことを言っているに過ぎない．この列の場合，ある分割の仕方をすれば条件を満たしてしまうのである（どんな分割か？）．

それでは選んだ列（$a^K b^K a^K b^K$）が悪かったのであろうか．分解できないことを示せればどんな列でも良いのである．残念ながらこれも否である．（理由：問題の言語 L に入る列は zz の形をしているのであるから，$u = w = y = \epsilon$, $v = x = z$ と置けば明らかに uv^iwx^iy は L に入る．）（解答終り）

要するにこの問題の言語が文脈自由言語でないことを言うためには定理 3.6 では不十分なのである．そこで，定理 3.6 をより強化してみよう．

定理 3.7　L を有限でない文脈自由言語とする．すると，L によって決まるある定数 K と K' が存在して，L に属す長さ K 以上の任意の列 z に対し，以下の 3 条件を満たす 5 個の列 u, v, w, x, y が存在する．*(i)* $z = uvwxy$ と書ける．*(ii)* $vx \neq \epsilon$．*(iii)* 任意の $i \geq 1$ に対して $uv^iwx^iy \in L$．*(iv)* $|vwx| \leq K'$.

証明　前の証明のキーポイントは導出木を見ていって，同じ変数が 2 回現れた（図 3.7 の X）ところをうまく利用している．ここで，部分列 vwx の長さは，この 2 回現れている変数のうちで上に出ている方の変数から下の部分木の葉の数である点に注意されたい．列 z の長さはいくら長くても良いので，最長パス上のこうした変数の繰り返しは各所に現れる可能性がある．それらを不用意に採用しては上の vwx の長さはいくらでも長くなってしまう可能性がある．正解は簡単で，一番長いパスを下から見ていって，同じ変数が始めて繰り返した部分を取れば良いのである．こうすれば，vwx の長さが文法で決まる（より正確には，変数の数と規則の右辺の長さの最大値で決まる）ある定数で押えられることが容易に分かる．（証明終り）

この強化された uvwxy 定理を使うと上の問題の言語が文脈自由言語でないことが証明できる．以前と同様に言語に入る列 $a^m b^m a^m b^m$ を考える．但し，値 m としては $m \geq K$ かつ $m \geq K'$ を満たす様な整数を選ぶ．問題の言語 L はそのような形のいくらでも長い列を含むので，そのような値 m は明らかに選べる．

この $a^m b^m a^m b^m$ を分割するのであるが，今度は $|vwx| \leq K' \leq m$ という条件がある．この条件を満たすためには，vwx の部分は，a または b の m 個の連続の内部か，$a^m b^m$ または $b^m a^m$ の境目に持ってくるしかない．しかし，そうすれば uv^2wx^2y が L に入らないことを示すのは簡単である．

3.6 まとめと文献

形式言語理論の研究の歴史を語るにはノーム・チョムスキーの業績に触れないわけにはいかない．チョムスキーは勿論著名な言語学者であるが，近年はむしろその（かなり過激的な）政治的発言がマスコミを賑わせている．特に 9.11 以降の発言は映画にもなった．

チョムスキーは 4 種類の文法を導入した．それらは，簡単な方から (i) 正規文法， (ii) 文脈自由文法， (iii) 文脈依存文法， (iv) ゼロ型文法の 4 種類で，最初の 2 つを本章で紹介した．実は正規文法については陽には記述していない．しかし，定理 3.3 の証明で与えた文法が正に正規文法なのである．より正確に言うと，生成規則の右辺が終端記号 a と変数 A に対して， a または aA の形をしている文法を正規文法と呼ぶ．

(iii) と (iv) の文法に関してもまもなく紹介する．これらは**チョムスキー階層**と呼ばれ，文法を段々強力なものにしていって表現できる言語の範囲を広げていくというごく自然な理論体系になっている．分類は細かすぎもせず，また大雑把過ぎもせず，全く見事な分類になっている．従って，計算時間や必要な記憶量を重視して分類する**計算量理論**が盛んになるまではこの分野のバイブルの様に扱われ，ほとんどの教科書がこのチョムスキー階層を説明することをその主要目的にしていた．本書においても，このチョムスキー階層の説明はその重要な目的のひとつになっている．チョムスキー階層が発表された論文は以下の論文である．

・N. Chomsky, “On certain formal properties of grammars,” Information and Control, 2,2, 137-167, 1959.

ここでちょっと注目してほしいのは論文が載った雑誌の名称である．この時代はいわゆる自動制御（automatic control）が華やかだった時代で，計算機に関する技術もともすればその中の一分野と見られないことも無かった．この雑誌は IEEE の論文誌と共に，計算機の理論に関する当時の重要な論文が発表される代表的な場であった．その後，この雑誌はその内容をどんどん計算機寄りに修正し，ついには 80 年代に雑誌の名称も Information and Computation に変更して，計算機の理論にとってより魅力的な雑誌を目指した．しかし，その結果はあまり芳しいとは言えないというのが個人的感想である．一方， IEEE の論文誌の方はというと，計算機の理論の論文に関しては，やはり昔日の面影は無くなっているようである．

ついでに言うと，「オートマトン（automaton）」という言葉も，私の学生時代（70 年代）ですらまだ一般的では無かった．ある論文を専門誌に投稿したときに，編集者校正ですべて「automation」に直って戻って来て唖然としたことを記憶している．学生諸君の試験問題の解答でも，「オートマント」や「オートマン」等々思わず吹き出してしまう記述があった．

この節で出典を断わるべき他の事項はグライバッハの標準形と uvwxy 定理であろう．前者は以下の論文に現れた．共に初学者にとっては易しい定理ではない．本書では特に前者の証明に関してはアイデアを述べるにとどめた．正確な証明を知りたい読者の方は（下の論文ではなく） 1 章で紹介した教科書を参照されることをお進めしたい．後者の定理は名前が面白くて有名な定理である．発表したのは Y. Bar-Hillel, M. Perles, E. Shamir の 3 人であるが，彼らの論文は 1961 年のドイツの雑誌に掲載されている（雑誌の具体的名称は省略する）．

・S. Greibach, “A new normal form thorem for context-free phase structure grammars,”

Journal of ACM, 12, 1, 42-52, 1965.

理解度の確認

問　言語 $\{0^n1^m \mid n = m \text{または} n = 2m\}$ を生成する文脈自由文法を求めよ．

問　言語 $\{wxw^{\mathrm{R}} \mid x, w \in \{0,1\}^*\}$ を生成する文脈自由文法を求めよ．（この問題はひっかけ問題である．注意！）

問　以下の言語はいずれも文脈自由言語ではないことを証明せよ．(i) $\{0^n1^{n^2} \mid n \geq 0\}$．(ii) $\{0^{2^n} \mid n \geq 0\}$.

問　正規言語の場合は和集合と共通集合の演算に対して閉じていた（つまり，言語 L_1 と L_2 が共に正規言語であるなら $L_1 \cup L_2$ と $L_1 \cap L_2$ は共に正規言語である）．文脈自由言語の場合は，和集合に対して閉じていることは練習問題 3.5 のところで述べた．しかるに，共通集合の演算に関しては閉じていないことを証明せよ．（ヒント：ある 2 つの文脈自由言語で，それらの共通集合が文脈自由言語でない様なものを発見できれば良い．「それらの共通集合」が $\{a^nb^nc^n \mid n \geq 0\}$ となるような 2 つの言語を探せ．）

問　言語 L が文脈自由言語であるとき，言語 $\{x^{\mathrm{R}} \mid x \in L\}$ もまた文脈自由言語であることを示せ．（ヒント：チョムスキーの標準形を使う．導出に必要なステップ数に関する数学的帰納法を利用する．）

問　練習問題 3.6 の文脈自由文法は $S \rightarrow SS$ と $S \rightarrow \epsilon$ という生成規則以外の生成規則全てに対して，以下の性質を満たしている．(i) 生成規則の右辺は終端記号で始まる．(ii) $A \rightarrow a\alpha$ と $A \rightarrow b\beta$ が左辺が等しい 2 つの生成規則であるなら $a \neq b$. 文法がこの性質を持っていると，目標の列が与えられた時，その列に対する最左導出を練習問題 3.6 のところで述べたようにほぼ自動的に求めることができる．これは大変良い性質で，例えばコンパイラ設計等で重要になってくる．上の問．．．の言語は上の 2 性質を持つ文脈自由文法では決して生成されないこと示せ．（ヒント：言語は 0^51^5 と 0^51^{10} の 2 つの列を含む．後者は長さ 10 のところまでは前者と全く同じである．従って，性質を満たす様な文法で生成しようとするなら，長さ 10 のところまでは，全く同じ規則が使用されるはずである．しかし，これはおかしい．）

問　言語に ϵ が含まれなければ，右辺が ϵ の生成規則は必要ないことを証明せよ．（導出木を考えたとき，ϵ に置き換えられる変数を上にたどって行くと，必ず枝分かれをする（つまり右辺が長さ 2 以上の）規則に行き当たる．その規則の変数を予め消しておくような規則を付け加えた上で，右辺が ϵ の規則を全て取り除く．）

問　文脈自由文法 G と終端記号からなる列 x が与えられた時，G が x を生成するかどうか調べるためにはどうしたら良いだろうか．（ヒント：一般性を失うことなく，G はチョムスキーの標準形と仮定して良い．その場合，x の長さが例えば 10 であるなら，x を導出するのに，右辺が 2 個の変数からなる規則を正確に 9 回使用しなければならない．その場合の数は有限であるから総当たりによって調べることができる．しかし，例えば左辺が 2 個の変数からなる規則が 5 種類あって目標の x の長さを n とおくと，この総当たりは真面にやると 5^n の場合を調べなければならない．勿論，5^n は n が増加するとすぐに膨大な数になってしまう．実はこれを n^3 程度の

ステップ数で実行する方法が知られているのである．まず

$$x = a_1 a_2 \cdots a_n$$

の ϵ でない部分列を (i,j) （$i \leq j$）で表す．例えば，$(5,7) = a_5 a_6 a_7$，$(4,4) = a_4$ である．このような部分列の総数は全体の列の長さが n であるから，およそ n^2 個程度である．次に部分列 (i,j) を生成できる変数の集合 $V(i,j)$ を求める．これは，$i = j$ の場合は簡単で，(i,i) は 1 個の終端記号（例えば 0）であるから，$A \to 0$ の形の規則の左辺の変数（2 個以上ありうる）を $V(i,i)$ に入れれば良い．次に $V(i,i+1)$ を求める．これは，$X \in V(i,i), Y \in V(i+1,i+1)$ である様な変数 X, Y に対して，$Z \to XY$ である様な変数 Z を全て入れれば良い．このことを続けていく．例えば，$V(4,8)$ を求めるには

$$X \in V(4,4),\ Y \in V(5,8),\ Z \to XY$$

$$X' \in V(4,5),\ Y' \in V(6,8),\ Z' \to X'Y'$$

$$\cdots\cdots$$

であるような，$Z, Z', \cdots$ を全て含める．最終的に，$V(1,n)$ が開始記号 S を含むなら列 x は導出される．）

問　定理 5.3 の証明で，ある変数からスタートして，右辺が長さ 2 の規則のみを使って（但し，最後にその先頭の変数を終端記号に置き換える）導出できる列の集合を正規表現で表したが，その正規表現はスター演算が 1 個だけの比較的簡単なものを例に取って説明した．以下ではそれがより複雑になったときのことを考えて見よう．例えば，変数 A に対する $V(A)$ が以下の様な正規表現であったとしよう．

$$V(A) = a(AB + C(AC + BC)^*)^*$$

事情があって，以下のような等価な正規表現に直す（理由はすぐに分かる）．

$$V(A) = a(AB + C + C(AC + BC)(AC + BC)^*)^*$$

ここでまず内側のスター演算に対して新しい変数 Z_1 を導入する．すなわち，

$$V(A) = a(AB + C + CZ_1)^*,\ V(Z_1) = (AC + BC)(AC + BC)^*$$

こうして，まず Z_1 を左辺に持つ規則が以下の様に導ける．

$$Z_1 \to AC,\ Z_1 \to BC,\ Z_1 \to ACZ_1,\ Z_1 \to BCZ_1$$

なお，上で元の式のままで，$V(Z_1) = (AC + BC)^*$ としてしまうと，$Z_1 \to \epsilon$ という規則が現れてしまう．これは，（後で解消できないことは無いが）面倒なので，上の様な正規表現の等価変換を行った．次に，A を左辺に持つ規則を以下の様に定める．

$$A \to a,\ A \to aZ_2,\ Z_2 \to AB,\ Z_2 \to C,\ Z_2 \to CZ_1,\ Z_2 \to ABZ_2,\ Z_2 \to CZ_2,\ Z_2 \to CZ_1Z_2$$

後は，以前の様に左辺が Z_1，Z_2 の規則に関して置き換えを行う．以上を一般化して説明せよ．

第 4 章

プッシュダウンオートマトン

正規言語と有限オートマトンでは明らかに性質が違った．すなわち，前者がどちらかというと言語に属する列を「生成」するイメージだったのに対し，後者は与えられた列が言語に入っているか否かを「判定する機械」と捉えることができた．この関係は前章の文脈自由言語と本章のプッシュダウンオートマトンの場合にも全く同じである．

4.1 プッシュダウンオートマトンの定義

オートマトンは動作する機械というイメージがあるため，その定義を数学的にきっちりと与えることは見掛け以上に困難である．この節では様相という概念を導入して上手にこの困難を乗り越える．実は筆者はこの定義を始めて見た時，一般性に富んだ実に見事な手法であると感激したことをおぼえている．読者の方もどうか味わって欲しい．

2.5 節で有限オートマトンの能力の限界を論じたとき，その直観的理由は内部の記憶状態が有限個にとどまっていたことであった．それでは，その状態を無限個にすれば良い．しかし，それをどのように記述するかという問題が残る．状態遷移図（例えば図 2.2 のような）で表そうとしても，異なった状態が無限に存在するのでは，書き下すことができない．（逆に言えば，有限の状態で色々興味深い無限言語を表現できるというのは，有限オートマトンの大きな利点である．）

オートマトンの世界では，状態を無限にするという目的のために，しばしばある特殊な「補助記憶」を導入する．補助記憶装置はその入出力動作がかなり制限されているので，有限の記述が自然に可能になり上の問題が回避できるのである．プッシュダウンオートマトン（長いので pda と省略する）の場合は，その名称が示唆する通り，補助記憶としてプッシュダウンスタック（これも以後単にスタックと略称）を使用する．スタックは図 4.1 の様な記憶装置で，以下の 3 種類の動作が許されている．(i) スタックの先頭の記号を読む（この時，その先頭に記号の下の情報については一切得ることができない）．(ii) スタックの先頭の記号を取り去る（この取り去る動作をポップと呼ぶことがある．先頭の記号を取り去った後は，自動的に 2 番目の記号が先頭に現れる）．(iii) 新たな記号をスタックの先頭に詰め込む（この動作をプッシュと呼ぶことがある）．(iv) スタックの内容には手をつけない．例えば図 4.1 の (a) の様に今スタックが記号列

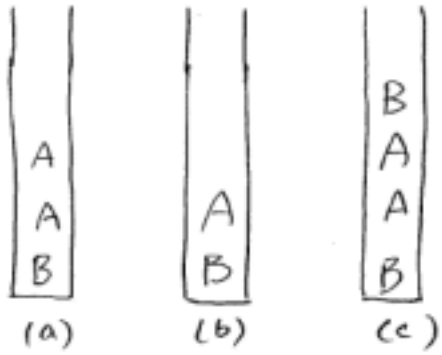

図 4.1 プッシュダウンスタック

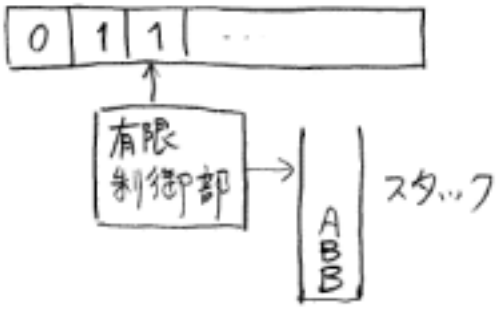

図 4.2 プッシュダウンオートマトン

AAB を保持していると仮定しよう（このように図の上方を列の左に，底を右に対応させる）．すると，上記 (i) を実行すれば情報として A が得られる．ここで注意すべきことは，見える（読める）のは先頭の記号（A）のみで，その下の記号に関しては何個あるか等も含めて一切の情報が得られない． (ii) のポップを実行すれば，スタックの記憶は同図 (b) の様に変化する．記号 B をプッシュすれば，同図 (c) の様に変化する．スタックには容量の制限がない．つまり，記号をいくらでもプッシュできる．この意味で，スタックは有限状態では記述できない．

pda は図 4.2 の様な構造である．入力テープと有限制御部は fa の場合と同じであるが，補助記憶として 1 個のスタックを有している． fa の場合を復習すると，その 1 ステップの動作は，(i) ヘッドの下の入力記号を読んでヘッドを 1 コマ右に移動させ， (ii) 現在の状態と (i) で読んだ入力記号によって決まる次の状態に遷移する，であった．同様に pda の 1 ステップ動作を記述すると以下の様になる． (i) ヘッドの下の入力記号とスタックの先頭記号を読んで入力ヘッドは 1 コマ右に移動する． (ii) それらの記号と現在の状態で決まる上記 (ii) または (iii) または (iv) のスタック演算を実行する． (iii) 同様に決まる次状態に遷移する．

fa では認識させることができなかった言語 $\{0^n1^n \mid n \geq 0\}$ に対して以下の様に動作する pda M を考えてみよう． M は記号 0 を読んだら，スタックに 1 個の記号 Z をプッシュする．記号 1 を読んだらスタックの先頭の記号をポップする．このような動作をさせるなら，読んだ 0 の個数と 1 の個数が同一の場合は，入力列を読み終った時に丁度スタックが 1 個の記号も無い状態になる（これを空になったという）．数が合っていないと，読み終る前にスタックが空になってし

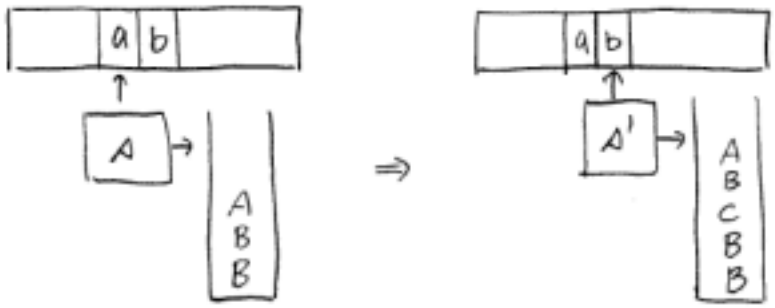

図 4.3 1 ステップの動作

まったり，あるいは読み終ってもスタックに記号が残っていることになる．以上， pda の動作を大体理解して頂いたと思うので，形式的定義を与えることにしよう．

定義 4.1　プッシュダウンオートマトン（pda）は $M = (K, \Sigma, \Gamma, \delta, s_0, Z_0)$ で与えられる．ここで， K， Σ， Γ はいずれも有限集合で，それらの要素はそれぞれ**状態**，**入力記号**，**スタック記号**と呼ばれる． $s_0 \in K$ は**初期状態**， $Z_0 \in \Gamma$ は**スタック初期記号**と呼ばれる． δ は**状態遷移関数**とよばれ， $K \times \Sigma \times \Gamma$ から $K \times \Gamma^*$ の有限部分集合族の中への写像である．

ややっこしいのは δ の定義で，これは pda の定義が初めから非決定性になっているためである．（初めから非決定性で定義する最大の理由は後で述べる文脈自由文法との等価性であるが，その他にも非決定性にしておかないと困る細かいことが色々出てくる．）仮に決定性であったとしたら，その定義は $K \times \Sigma \times \Gamma$ から $K \times \Gamma^*$ の中への写像となっていたはずで，その場合，例えば

$$\delta(s, a, A) = (s', ABC)$$

という感じの写像になる．これは，もちろんこれは 1 ステップの動作を表していて，この場合，現在の状態が s で，入力記号として a を読み，スタックの先頭に記号が A なら，次状態は s' で，スタックの先頭の記号（今の場合は A）をスタック記号の列 ABC で置き換える（図 4.3.1 参照）という動作を指定する．ここで，「スタックの先頭記号を置き換える」という動作が以前導入したスタックの動作と違うではないかと思われるであろう．しかし，例えば， (iv) の「何もしない」という動作は先頭の記号をそれと同じ記号で置き換えるという動作， (ii) のポップは ϵ で置き換えるというという動作で代用できる．また， (iii) のプッシュは例えば先頭の記号 A の上に B をプッシュするのなら A を BA で置き換えるという動作で代用できる．つまり，上の定義で前述のスタック動作は全て記述できるのである．実はこの定義はもっと広い．つまり，図 4.3.1 の様に，先頭の記号 A を記号列 ABC で置き換えるということまで可能にしている．これは，前述の約束なら，まず A をポップして，次に記号 C， B， A をこの順にプッシュするという合計 4 ステップの動作が必要になるのに， 1 ステップで可能にしてしまっている．このことは，詳しく説明すると結構長くなるのであるが，まあここではこれが正式な定義であるとして受け入れていただきたい．

次に非決定性の説明をしよう． fa の場合と同じように δ の右辺が集合になる．例えば，

$$\delta(s, a, A) = \{(s', ABC), (s'', \epsilon)\}$$

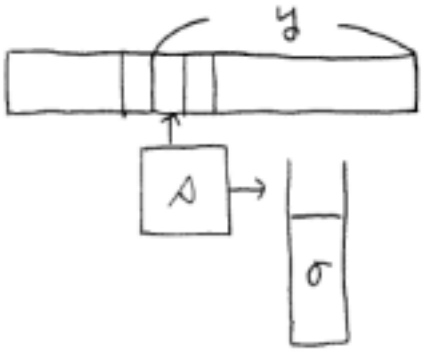

図 4.4 様相

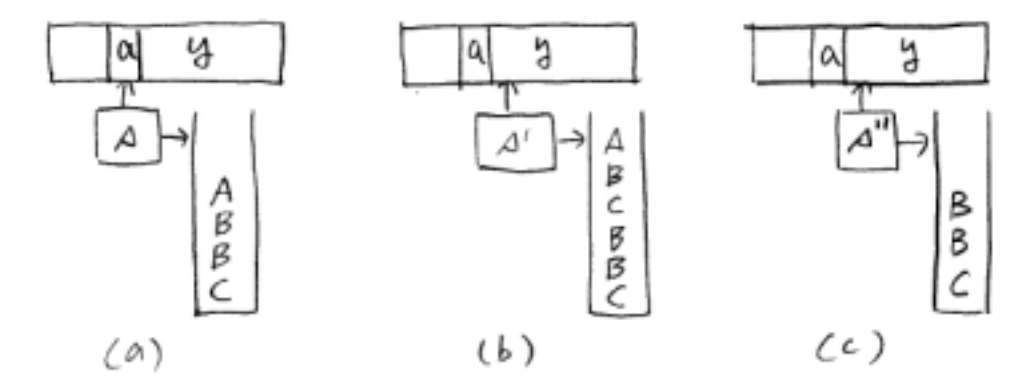

図 4.5 様相の 1 ステップ変化

といった感じである．解釈は，これも fa の場合と同様で，第 1 の動作も第 2 の動作もいずれも可能であると考える．以上で pda そのものの定義は終了であるが，次にそれが認識する言語の定義をしなければならない．

定義 4.2　$s \in K$ を状態，$y \in \Sigma^*$ を入力記号の列，$\sigma \in \Gamma^*$ をスタック記号の列としたとき，これらの三つ組み (s, y, σ) を**様相**と呼ぶ．

この定義はまことに味気ないが，それは動作途中における pda M の状況（スナップショット）を表していると考えて欲しい．つまり，今 M の様相が (s, y, σ) であるといったときは，M が図 4.4 の様な状況になっていることを意味する．つまり，M はすでに入力テープの一部を読み終わっているが，まだ列 y を読み残している．また，スタックの内容は列 σ で，有限制御部の状態は s である．既に読み終わったテープの部分は無視してしまうということに違和感を感じられるかも知れないが，読み終わった部分は今後の動作に影響を及ぼさない．つまり，様相が与えられえればこの後の動作が一意に定まるのである．

例えば M の現在の状況が図 4.5(a) の様になっていて，状態遷移関数は上と同じ $\delta(s, a, A) = \{(s', ABC), (s'', \epsilon)\}$ であったと仮定しよう．すると，1 ステップ後の状況は，同図 (b) の様に変わりうる．これを様相を使って，様相 $(s, ay, ABBC)$ は様相 $(s', y, ABCBBC)$ に **1 ステップで変わりうる**と呼ぶことにする．（同図 (c) の様にも変わりうるので，様相 $(s, ay, ABBC)$ は様相 (s'', y, BBC) にも 1 ステップで変わりうる．）この「1 ステップで変わりうる」というのはより数学的には 2 つの様相間の関係として定義することができる．これについては，ちょっと面倒な

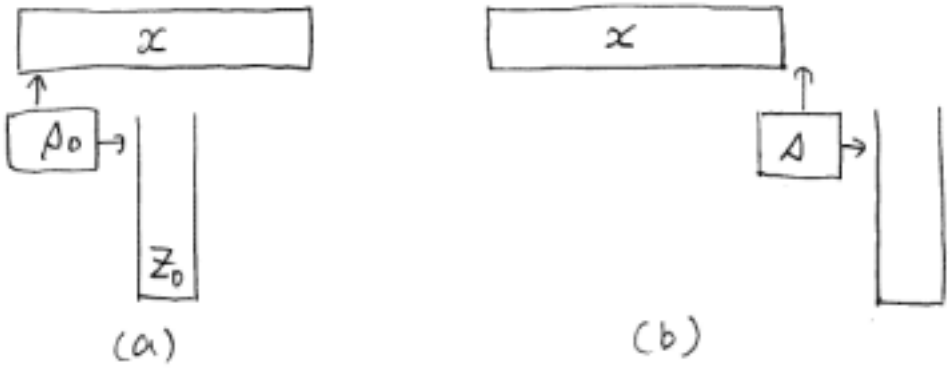

図 4.6 初期様相と受理様相

ので章末の問題を参照していただくことにして，以下ではその記法のみ定義することにする．

定義 4.3　様相 S_1 から S_2 へ *1* ステップで変わりうるとき $S_1 \Rightarrow S_2$ と書く．

上の例の場合なら，

$$(s, ay, ABBC) \Rightarrow (s', y, ABCBBC)$$

$$(s, ay, ABBC) \Rightarrow (s'', y, BBC)$$

である．

定義 4.4　$S_1 = S_n$，又は $S_1 \Rightarrow S_2$，又は $S_1 \Rightarrow S_2 \Rightarrow \cdots \Rightarrow S_n$ であるような *1* 個以上の様相 $S_2, \cdots S_{n-1}$ が存在するとき，様相 S_1 は S_n へ**何ステップかで変わりうる**と呼び，$S_1 \overset{*}{\Rightarrow} S_n$ と書く．

定義 4.5　列 $x \in \Sigma^*$ がある状態 $s \in K$ に対して条件

$$(s_0, x, Z_0) \overset{*}{\Rightarrow} (s, \epsilon, \epsilon)$$

を満たすとき，x は $pdaM$ によって**受理される**という．受理される列の全体を M によって**認識される言語**と呼ぶ．

上の受理される列の定義を解説しよう．条件の左側の様相はいわゆる初期様相で，図 4.6(a) の様に表すことができる．ここで，スタックの内容はスタック初期記号 1 個からなっていることに注意されたい．空のスタックから始めると色々と都合が悪い．例えば，δ の定義も（最初は読むべきスタックの先頭の記号が存在しないので）変更しなければならない．スタック初期記号を導入した理由を理解して欲しい．右側の様相は受理様相で，同図 (b) の様に書ける．ここでは，テープを読み終わっていることとスタックが空になっていることに注意して欲しい．そのときの状態は何でもよい．つまり，テープを読み終わる最後のステップの後でスタックが空になるとき受理されるのである．有限オートマトンの時は受理状態を導入し，受理状態で終わるときに受理されると定義した．こちらの方が自然であると思われるかもしれない．上のようなちょっと奇妙な受理条件がなぜ採用されたかは，しばらく後で pda と文脈自由文法の関係を論じるときに分かるであろう．

ようやく退屈な定義が終わった．前にも述べた様に，この定義はかなり巧妙にできているので，あえてできるだけ数学的厳密さを失わないように丁寧に記述したつもりである．さて，例題を見てみよう．

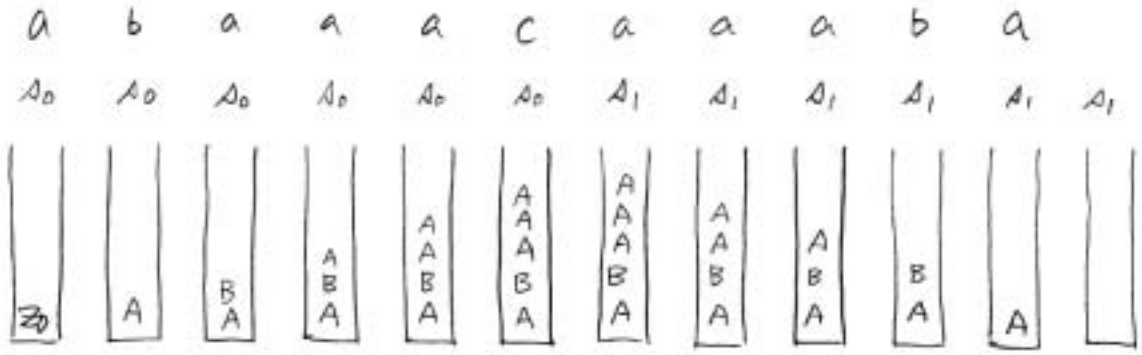

図 4.7 プッシュダウンオートマトンの動作例

練習問題 4.1　$\{xcx^R \mid x \in \{a,b\}^*\}$ を認識する pda を構成せよ．

解答　状態集合は $\{s_0, s_1\}$ で，s_0 が初期状態，スタック記号は $\{A, B, Z_0\}$ を使用する．直観的には列を左から見ていって，c が現れるまでスタックに詰め込んでいく（ただし，入力記号 a に対して，スタック記号 A，b に対して B を対応させる）．注意すべきことは，先に読んだ（つまり左の方の）記号がスタックのより底の方に入っていくことである．例えば，aab を読めばスタックは先頭から BAA になっている．c が現れたら今度は動作モードを変えて，1 文字づつポップしながらスタックの先頭の記号と一致するかをチェックしていく．上の場合なら後半の入力列が baa ならこのチェックが成功することになる．以下が状態遷移関数の書き下しである．

$$
\begin{array}{lcl@{\qquad}lcl@{\qquad}lcl}
\delta(s_0,a,Z_0) & = & \{(s_0,A)\} & \delta(s_0,a,A) & = & \{(s_0,AA)\} & \delta(s_0,c,A) & = & \{(s_1,A)\} \\
\delta(s_0,b,Z_0) & = & \{(s_0,B)\} & \delta(s_0,b,A) & = & \{(s_0,BA)\} & \delta(s_0,c,B) & = & \{(s_1,B)\} \\
\delta(s_0,c,Z_0) & = & \{(s_1,\epsilon)\} & \delta(s_0,a,B) & = & \{(s_0,AB)\} & \delta(s_1,a,A) & = & \{(s_1,\epsilon)\} \\
 & & & \delta(s_0,b,B) & = & \{(s_0,BB)\} & \delta(s_1,b,B) & = & \{(s_1,\epsilon)\}
\end{array}
$$

よく見ると，例えば $\delta(s_1,a,B)$ は書かれていない．このような書かれていない組み合わせに対する δ の値は ϕ（空集合）であると（これ以降も）考えて欲しい．また，上の δ の右辺は集合の形をしているとは言え，全て 1 個の要素から成っている．即ち，この pda は実際は決定性の動作をしていることになる．

さて，例えば列 $abaaacaaaba$ に対するこの pda の動作を図 4.7 に示した．図の見方は説明の必要が無かろう．例えば，最初は入力ヘッドを左端の a の上，初期状態 s_0，スタックの列 Z_0（つまり様相 $(s_0, abaaacaaaba, Z_0)$）からスタートする．最初のステップは $\delta(s_0,a,Z_0) = \{(s_0,A)\}$ によって決まり，Z_0 が A に置き換えられ様相 $(s_0, baaacaaaba, A)$ に移る．状態 s_0 が上で述べた前半の動作モード（スタックに積み上げる）を担い，s_1 が後半のチェックモードを担う．上で述べた $\delta(s_1,a,B)$ はチェックが失敗したときに現れる動作であるから，右辺が空集合（つまり次の様相が無い）になっていても不思議ではない．上の列の場合はチェックが全て成功し，最後のステップの後は確かに受理様相 (s_1,ϵ,ϵ) になって受理する．

xcx^{R} ではない形の列を受理しないことも示さねばならない．こちらの方はちょっと厄介である．図 4.8 に $abaaacaabba$ と $abaaacaaa$ の場合を示した．前者の場合は途中で右辺が空集合の δ を適用することになって，この場合次の様相は存在しない．後者の場合は，テープを読み終わっ

図 4.8 非受理の動作例

た時のスタックが空になっていない．これもその次の様相が存在しないので，受理様相に到達することは明らかに不可能である．（解答終わり）

練習問題 4.2　$\{xx^R \mid x \in \{a,b\}\{a,b\}^*\}$ を認識する pda を構成せよ．

解答　一見似た問題であるが実は大いに異なる．以前の問題では存在した列の真ん中を示す記号 c がなくなっていることに注意されたい．基本的アイデアは前の問題の場合と同じで，前半が積み上げモード，後半がチェックモードになる．ただ，前半から後半に移るべきタイミングが，以前の例題の場合なら記号 c を見て移行すればよかったのであるが，今回はこの c が存在しないのである．この言語を認識する pda の状態遷移関数は以下の様になる．

$$
\begin{array}{rclrclrcl}
\delta(s_0,a,Z_0) & = & \{(s_0,A)\} & \delta(s_0,a,A) & = & \{(s_0,AA),(s_1,\epsilon)\} & \delta(s_1,a,A) & = & \{(s_1,\epsilon)\} \\
\delta(s_0,b,Z_0) & = & \{(s_0,B)\} & \delta(s_0,b,A) & = & \{(s_0,BA)\} & \delta(s_1,b,B) & = & \{(s_1,\epsilon)\} \\
 & & & \delta(s_0,a,B) & = & \{(s_0,AB)\} & & & \\
 & & & \delta(s_0,b,B) & = & \{(s_0,BB),(s_1,\epsilon)\} & & &
\end{array}
$$

以前と同様，列 $abaaaaaaba$ に対する動作を図 4.9 に示す．今回は δ の右辺が 2 個の動作の集合になっている場合があるので，真に非決定性である．従って，可能な動作はこれ以外にも多数存在する．例えば左から 5 番目の a の上にヘッドがあるときの様相は $(s_0, aaaaba, AABA)$ であって，$\delta(s_0,a,A) = \{(s_0,AA),(s_1,\epsilon)\}$ が適用される．最終的に受理様相にたどり着くのは，右辺の第 1 の動作を選択した場合で，もし第 2 の動作を選択すれば，図に示される様に次の様相は $(s_1, aaaba, ABA)$ となる．更に 1 ステップは動作するが，その時点で次の様相が存在しないことになってしまう．

図に示した受理に至る動作は列の前半を読み終わるまでは常に s_0（つまり積み上げモード）に留まり，後半の最初の a を読んだときに，(s_1,ϵ) の方の動作を選択してチェックモードに移行している．チェックモードに移行した後は完全に決定性の動作を行うので，例えば上で述べたような中央より以前でチェックモードに移行してしまった場合は受理様相には到達できないことが分かるであろう．しかし，定義によれば，受理様相に至るような様相の系列が唯ひとつでも存在すればよいのであるから，このような（失敗の）動作系列が存在してもかまわない．

受理してはいけない列，例えば，$abaaaaabba$ の場合はどうであろうか．上で述べた様に，正しい場所でチェックモードに移行した場合は右から 3 番目の記号 b のところで前の問題と同じように右辺が空集合の δ に当たってしまう．それ以外の場所でチェックモードに移行した場合も丹

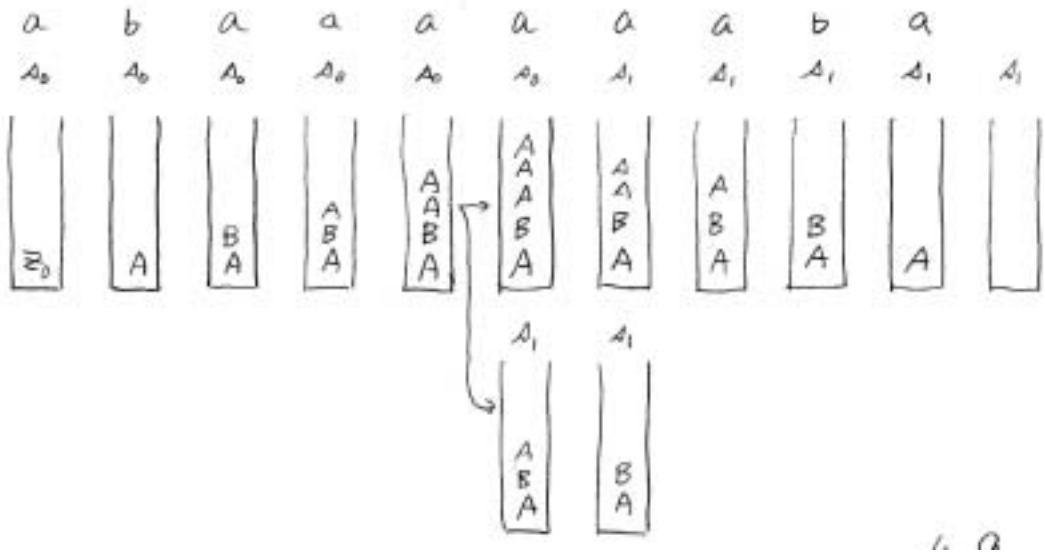

図 4.9 非決定性の動作例

念に調べればいずれもどこかで次の様相が存在しないような（受理様相でない）様相に突き当たってしまうことが分かる．従って，受理しない．（解答終わり）

以前の文脈自由文法の場合と比べて，（特に非決定性が絡んだ場合は）説明がかなり困難であることを感じ取っていただけたであろうか．このような状態遷移関数を書き下す説明は，この程度の機械でもこの有り様で，機械がもっと複雑になってくると明らかに破綻してしまう．それに，状態遷移関数を書き下すということ自体がかなり面倒くさい．上の例でも，機械の動作は直観的にはそれほど複雑では無いにもかかわらず，大変な労力を必要としている．

4.2 プッシュダウンオートマトンの設計

例えば試験である言語を認識する pda を設計せよという試験問題が出たとしても，上のような状態遷移関数を書き下すことはほとんどの場合しないのである．限られた時間では到底不可能であるし，仮に書き下せたとしても，そのまま解答とされたのでは，真偽の判定に膨大な時間がかかってしまい，今度は採点者が困ってしまう．

それでは pda の設計とは何をすることなのであろうか．それは，一言で言えば，「意中の pda の動作を上手に分かりやすく説明する」ことなのである．その動作を形式的に実現するのが状態遷移関数であるから，動作が必要にして十分な詳しさで説明できていれば，あとの状態遷移関数に落す作業はまあ時間さえかければ何とかなるであろうと考えるのである．実はこの「必要にして十分な詳しさ」が難しいところで，これは経験を積んで頂くしかない．この作業は実に妙味のあるところで，筆者は大好きである．読者の方にとっても色々な意味で良い訓練と経験になると信じている．

それでは，前節の練習問題 4.2 の pda を M とおいて，その動作を説明して見よう． M の動作には 2 つのモード，積み上げモードとチェックモードがある．最初は積み上げモードからスタートする． 1 ステップの動作は以下の様になる．

（1）積み上げモードの動作．現在読んでいる記号が列の中央の丁度右側の記号（つまり，入力列が $abaaaaaaba$ の場合は右から 6 番目の a）かどうかをゲスする．そうでないとゲスすれば，M は読んでいる記号をスタックにプッシュし，積み上げモードを続ける．そうであるとゲスすれば，スタックの先頭記号と読んでいる入力記号が同一であるかどうかを調べ，同一ならポップし，チェックモードに移行する．もし同一で無ければ M はそこで停止する．（ゲスは guess である．後で解説する．）

（2）チェックモードの動作．M は現在読んでいる記号とスタックの先頭の記号を比較し，異なれば停止．同じならポップしてチェックモードを続ける．

以上で動作の説明終了である．キーポイントは（1）のゲスである．読者諸君が M になったつもりで考えて欲しい．M が一番知りたいのは，入力テープの中央の場所である．しかし，左から順々に（a または b を）読んでいく限りにおいてはこのことは知りようがない．そこで，「ゲス」をするのである．ここでは「特に理由無く推測する」という意味で使っている．定義にはこんなゲスなどというものは一切出てこなかったと仰有るかもしれない．その通りなのであるが，もう少し我慢して読んで欲しい．

このゲスを状態遷移関数にどのように反映させるかであるが，これは単に非決定的選択を対応させるのである．つまり，状態遷移関数の右辺を 2 つ（以上）の動作の集合にしてしまえば良い．上の例なら，$\delta(s_0, a, A) = \{(s_0, AA), (s_1, \epsilon)\}$ の様にである．そうすると，列を受理するための定義は以下の様に言い換えることができることを分かって欲しい．つまり，「与えられた列は，受理様相に至る様なゲスの系列が存在する時，またその時に限って受理される」となる．上の例なら，$abaaaaaaba$ に対しては，最初の a に対しては，中央の右では「ない」とゲスし，次の a に対しても「ない」とゲスし，という様に 5 番目の a まで，「ない」とゲスし，6 番目の a で「ある」とゲスすれば最後にスタックが空になって受理様相に到達する．

ここで注意して頂きたいことは，上での新しいゲスを使った受理条件では個々のゲスが正しいか間違っているかについては一切言及していないことである．これは，上の状態遷移関数への反映から見ても，2 つ（以上）のゲスの結果が対等に扱われているので，当然のことである．ただ，ゲスは当然何かを推測するので，それが当っているときに受理に到達するという説明にした方が分かりやすい．上の例なら，列の中央の場所をゲスして，全てのゲスが当った時に受理する様に設計している．ゲスがはずれた時に動作はどのように考えたら良いのであろうか．これも上の新しい条件に照らしてみれば，現在の列が受理すべき列であるなら（当然正しいゲスによって受理されるなずであるから）「どうでも良い」としか言いようがない．受理すべきでない列であるなら，（ゲスが当れば当然受理しないように設計されているはずであるが），ゲスがはずれた時も絶対に受理してはいけない．更に例を見て頂いて理解して欲しい．

練習問題 4.3　$\{0^n 1^m \mid n \neq m\}$ を認識する pda を構成せよ．

（解答）pda M は最初に $n > m$ であるか $n < m$ であるかをゲスする．（もし，$m = n$ であればこのゲスは絶対にはずれるが問題ない）．$n > m$ とゲスすれば，最初の 0 を無条件で読み飛ばして記号ゲスモードに入る．（読み飛ばすとは，スタックに何もしないで入力ヘッドを右に動かす動作のことである．）記号ゲスモードでは，現在読んでいる 0 が左から $n - m + 1$ 番目の 0 であるかどうか（つまり，図 4.10 で矢印で示されているような場所の 0 であるかどうか）をゲスする．yes とゲスすれば，チェックモードに入る．no とゲスすれば記号ゲスモードを続ける．チェックモードでは 0 を読めばそれをスタックにプッシュし，1 を読めばポップする．こ

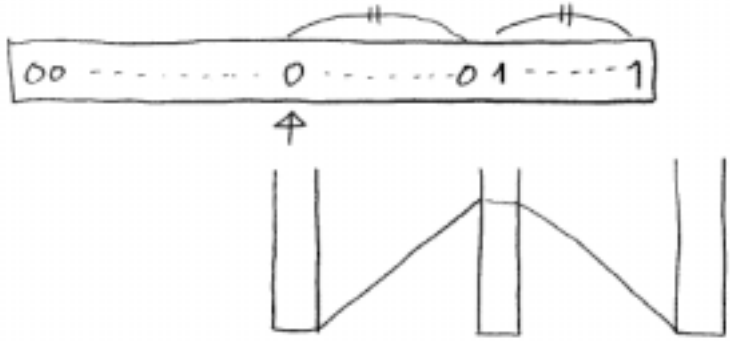

図 4.10 記号ゲスモードとチェックモード

の過程で，例えば 1 の後に 0 が来る等という列の形がおかしいことが分かったらその場で停止して（列を読み終っていないので当然）非受理とする．

$n < m$ とゲスした場合は，積み上げモードに入って，全ての 0 をスタックに積み上げる．次に 1 を読んだときは，最初の 1 個の 1 は無条件で読みとばした後で，記号ゲスモードに入る．ここでは読んでいる 1 が左から $m - n + 1$ 番目の 1 であるかどうかをゲスする． yes とゲスしたらその 1 からポップモードにはいって， 1 を読む度にスタックの記号をポップする． no とゲスしたら記号ゲスモードを続ける．

正しさを検証しよう．与えられた列が言語に入っているとしよう．すると，必ず $n > m$ か $n < m$ であるから，最初のゲスはどちらかが必ず当る． $n > m$ で，且つ最初のゲスが当ったとしよう．すると，図 4.10 の様な場所の 0 は必ず存在するので，その場所で始めて yes とするような記号ゲスモードでの正しいゲスの系列が存在する．この場合に読み終った後でスタックが空になっていることはほとんど明らかであろう． $n < m$ の場合も同様である．

次に与えられた列が言語に入っていないとしよう．まず，列の形が $0^n 1^m$ の形になっていない場合は上で述べたように途中で停止し受理しない．列の形が正しい場合は $n = m$ である．最初に $n > m$ とゲスしたとしよう．すると，最初の 0 を読み飛ばしているので，どこでチェックモードに入ったとしても，それから後にスタックに積み込める 0 の数は後の 1 の数よりも少ない．従って， 1 を読み終る前にスタックが空になって，そこで停止してしまうので受理しない． $n < m$ をゲスした場合も同様である． （解答終り）

次の例題を見る前に， pda で有限オートマトンを模倣できることを知っておいてもらいたい．図 4.1 に示したように pda は fa に補助記憶を追加したものであるから， fa を模倣できるのは当たり前であると思われるかも知れない．それは正しいのであるが， 1 点問題点があって，それは受理条件である． pda は，受理する場合は，ちょうど入力を読み終わったときにスタックを空にしなければならないのである．これは今読んでいる記号がテープの右端であることが分かれば簡単に実現できるが一般には分からない．

解決策はゲスを利用することである．（決定性）有限オートマトン M が与えられたとする． M を模倣する pda T は M の状態遷移を，スタックに一切手をつけることなく（つまり最初の記号 Z_0 をそのままにして）模倣しながら，現在の記号がテープの右端であるかどうかをゲスす

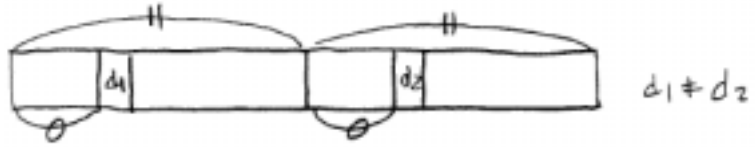

図 4.11 列の構造

る． yes とゲスして，その記号を読んだ行き先が M の受理状態ならスタックから Z_0 をポップする． no とゲスしたら（スタックはいじらずに）普通に模倣を続ける．正当性はほぼ明らかである．もし， pda が決定性であったなら，（今の様な受理条件のもとでは） fa さえもシミュレートできないことに注意されたい．

練習問題 4.4 $\{x \mid x \in \{a,b\}^* \text{かつ} x = yy \text{とかけない}\}$ を認識する pda を構成せよ．

（解答）言語に入っている列 x は，以下の条件 (i) か (ii) のいずれかを満たすことが分かる．(i) 長さが奇数である． (ii) 長さが偶数で，

$$x = a_1 a_2 \cdots a_n a_{n+1} \cdots a_{2n}$$

としたとき，ある整数 i に対して， $a_i \neq a_{n+i}$．（図 4.11 参照．この様な異なった 2 個の記号を d_1， d_2 とおく．）以下の様に動作する pda M を構成すればよいであろう．

（1） M は最初に上の (i) または (ii) のいずれの条件が満たされるのかをゲスする． (i) とゲスした場合は，その後は奇数チェックモードに入り，上で述べた様に fa の模倣によって本当に長さが奇数であるかどうかをチェックする．奇数であったなら受理するし，そうでなければ受理しないで停止．

（2） (ii) とゲスしたら次に記号 d_1 をゲスする．テープの最初からそのゲスした d_1 までの距離（記号数）をスタックに蓄える（図 4.11 参照）．次にヘッドを右に動かしながら（スタックはそのままで）テープの中央をゲスする．ゲスされた中央からはスタックの内容を 1 個づつポップしていくと d_2 の場所が分かるので，その記号が以前に記憶しておいた d_1 と確かに異なっていれば，残りの記号を読み飛ばして受理する（上の fa の模倣の様に最後の記号をゲスしてスタックの記号をポップする）． d_1 と同じであったらその場で停止して受理しない．ここで，念のため，上の d_2 の場所の決め方をより詳しく説明する．例えばゲスされた d_1 が先頭から 5 個目の記号だったとする． d_1 より左の 4 個の記号に関して，それらを 1 個読む度に 1 個の記号 A をプッシュするので， d_1 を読み終った段階でスタックの記号列は $AAAAZ_0$ となっている． Z_0 をスタックの底を示すものとして残すことに注意されたい．テープの中央を越えたあとは， 1 個の入力記号を読む度に 1 個の A をポップする．こうして， Z_0 がスタックの先頭に現れた時のヘッドの下の記号が d_2 である．

いかがであろうか．残念ながらこの答えは間違いである．例えば，図 4.12 の様に d_1 と中央をゲスすると，受理してしまう．これは明らかに受理してはいけない列で，反例になっている．失敗の原因は何であったのだろうか．

上の動作はゲスが当たった時は全く問題ない．列が言語に入っているかどうかによって，正しく受理・非受理の決定を行う．問題はゲスが外れた場合である．よく考えてみれば，ゲスが外れ

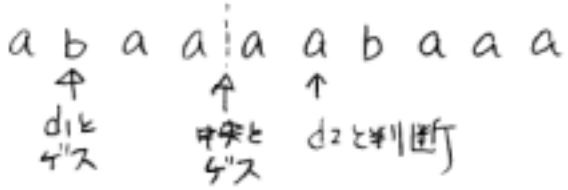

図 4.12 反例

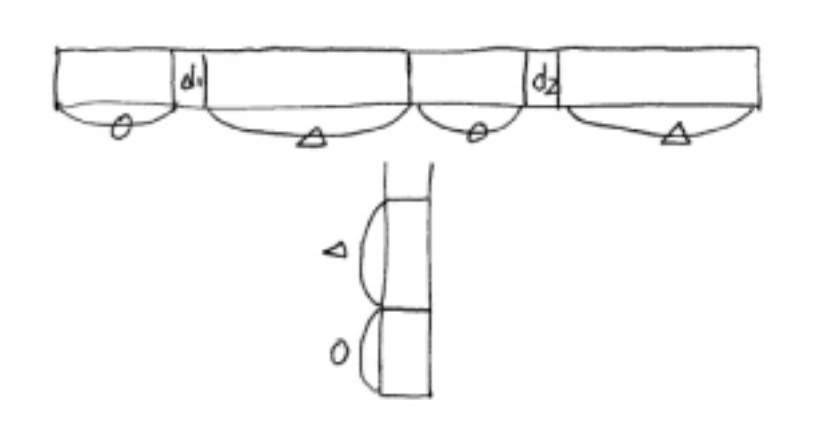

図 4.13 設計の修正

たときは常に非受理にしておくのが安全である．実は今までの例題でも常にそうしているのである．例えば，前問で $n > m$ とゲスしたのに実際は $n < m$ であったなら，受理しないように設計してあることを再確認して欲しい．この問題でも，最初の (i) または (ii) のゲスで，(i) とゲスしたのに実際は長さが偶数ならやはり受理しないようになっている．上の反例はゲスが外れたときにも受理する可能性が残っていたから生じたと考えることができるのである．

それではゲスが外れたときには受理しないように動作を修正しよう．しかし，このことができるためには，ゲスが外れたことを機械自身が知ることができなければならない．今の場合テープの中央をゲスしているので，そのゲスが正しいかどうかをチェックするためには，何らかの方法で，ゲスするまでに読んだ記号数を全てスタックに蓄える必要がある．例えば，図 4.13 の様に d_1 の前の記号数と後の記号数を違った記号でスタックに蓄えることは可能である．しかし，残念なことに，この場合，d_1 の後の記号数がスタックの上に入っている．中央とゲスした後で（d_2 の場所を決めるために）必要なのは d_1 の前の記号数なのであって，これが d_1 の後の記号数によって隠されてしまっている．つまり，これではだめなのである．勿論，このような記号数の上下関係を逆にできれば理想であるが，スタックの性質からこれは望みが無い．

図 4.13 を見ているうちに優秀な読者の諸君は気が付かれたのではないかと思うが，実は中央のゲスは必要ないのである．記号 d_1 と d_2 が正しい位置関係にあるための必要十分条件は，図 4.14 の様に，d_1 の前，d_1 と d_2 の間，d_2 の後の記号数をそれぞれ l_1，l_2，l_3 としたとき，単に

$$l_2 = l_1 + l_3$$

が満たされれば良いのである．このことをチェックするのは，同図に示すようにそれほど困難ではない．結局，以下の様に動作する pda M で認識することができる．

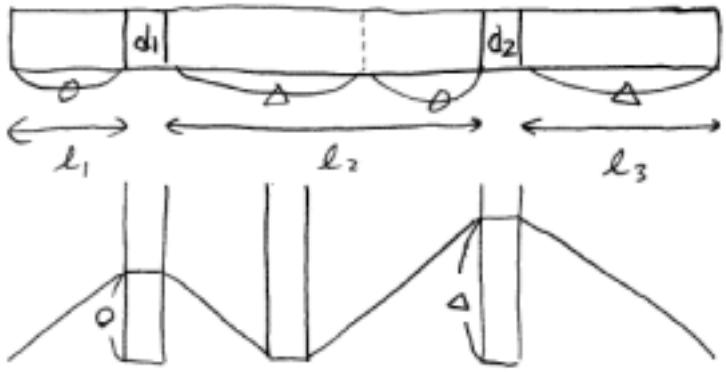

図 4.14 正しい構成

M は現在の記号が d_1 かどうかをゲスする． no とゲスすれば，１個の記号をプッシュして，次に進む． yes とゲスすればその記号が 0 であるか 1 であるかを有限状態を利用して記憶し（章末問題），次に d_2 ゲスモードに入る．そこでは，まず，スタックの記号を 1 個づつポップしながら入力記号を読み飛ばす．スタックが空になったら（本当に空にしてはそこで停止してしまう．最初にある Z_0 を底を示す記号として利用すればよい），今度は d_1 をゲスしたのと全く同様にして（つまり 1 記号をプッシュしながら） d_2 をゲスする． yes とゲスしたら記憶している d_1 の値と確かに違っているかを調べる．もし，同じであったらその場で停止する．違っていれば，排出モードに入り， d_2 の右の記号を 1 個読むごとにスタックの記号を 1 個ポップする．正当性に関しては読者自身で証明して欲しい．（解答終わり）

4.3 文脈自由文法との等価性

プッシュダウンオートマトンが重要である最大の理由は，その文脈自由文法との等価性であると言って過言でない．文脈自由文法に合わせるために，スタックという補助記憶装置を導入し，さらに受理条件も奇妙なものを採用しているのである．更には非決定性を標準にしている．この節でこの等価性の証明を行うが，それは以前の正規文法と有限オートマトンの等価性の証明よりも格段に困難である．

準備として， pda の場合は有限状態が「必要ない」ことを示す．必要ないとは，つまり，１個の状態で十分であるということである．但し，スタック記号に関しては制限していないことに注意して欲しい．つまり，基本的アイデアはスタック記号に有限状態の情報を押し込んでしまうことである．

練習問題 4.5 $\{xx^R \mid x \in \{a,b\}^*\}$ を認識する *1* 状態の *pda* を構成せよ．

（解答）以前の問題 4.2 の解答では 2 個の状態 s_0 と s_1 を使用して，積み上げモードとチェックモードを区別していた．今回は，スタックの先頭記号をみればそれらの違いが分かるようにする．そのために，積み上げモードでは先頭の記号が以前の A または B であったのを A' または B' になるようにする．勿論，先頭より下の部分については従来通り， A または B を使用する．そうすれば先頭の記号をポップすることが自動的にモードの変更になる．念のために状態遷移関

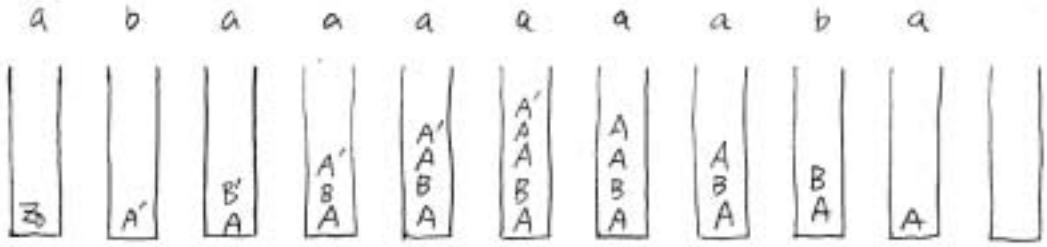

図 4.15 1 状態による模倣

数を以下に書き下して，その受理様相に至る動作例を図 4.15 に与える．

$$
\begin{array}{rclrclrcl}
\delta(s_0,a,Z_0) & = & \{(s_0,A')\} & \delta(s_0,a,A') & = & \{(s_0,A'A),(s_0,\epsilon)\} & \delta(s_0,a,A) & = & \{(s_0,\epsilon)\} \\
\delta(s_0,b,Z_0) & = & \{(s_0,B')\} & \delta(s_0,b,A') & = & \{(s_0,B'A)\} & \delta(s_0,b,B) & = & \{(s_0,\epsilon)\} \\
 & & & \delta(s_0,a,B') & = & \{(s_0,A'B)\} & & & \\
 & & & \delta(s_0,b,B') & = & \{(s_0,B'B),(s_0,\epsilon)\} & & &
\end{array}
$$

正当性に関しては各自が確認して欲しい．後でより形式的に述べるが，このように有限状態を無くしてしまうと，それはほとんど文脈自由文法と同じなのである．この例の場合なら以下の様な生成規則の集合を与えればよい．

$$
\begin{array}{lll}
 & A' \to aA'A, \quad A' \to a & \\
Z_0 \to aA' & A' \to bB'A & A \to a \\
Z_0 \to bB' & B' \to aA'B & B \to b \\
 & B' \to bB'B, \quad B' \to b &
\end{array}
$$

図 4.15 に与えた動作に対応する導出は以下のようになる．

$$
\begin{aligned}
& Z_0 \Rightarrow aA' \Rightarrow abB'A \Rightarrow abaA'BA \Rightarrow abaaA'ABA \Rightarrow abaaaA'AABA \\
& \Rightarrow abaaaaAABA \Rightarrow abaaaaaABA \Rightarrow abaaaaaaBA \Rightarrow abaaaaaabA \Rightarrow abaaaaaaba
\end{aligned}
$$

このように，個々の生成規則は状態遷移関数の個々の書き下しに完全に一対一に対応しているし，更に導出の各ステップも pda の各ステップの動作に完全に一対一対応している．このように， pda の状態を無くしてしまえば，それはもう文脈自由文法とほとんど同じであることが分かって頂けたと思う．（解答終り）

上の例は特別な場合であるが，実はこの状態を取り去るという作業は常に可能なのである．

定理 4.1　与えられた *pda* に対して，同じ言語を認識する状態数 *1* の *pda* が構成できる．

（証明）形式的に証明しようとすると大変なので，ここではそのアイデアを述べるにとどめる．図 4.16 を参照されたい．与えられた（多状態の） pda を M とする．図に示すように， M が入力記号 a， b， c を，状態を q， p， r と変化させながら読み，その間にスタックに B をプッシュし，直後にそれをポップするような動作を行ったとしよう．

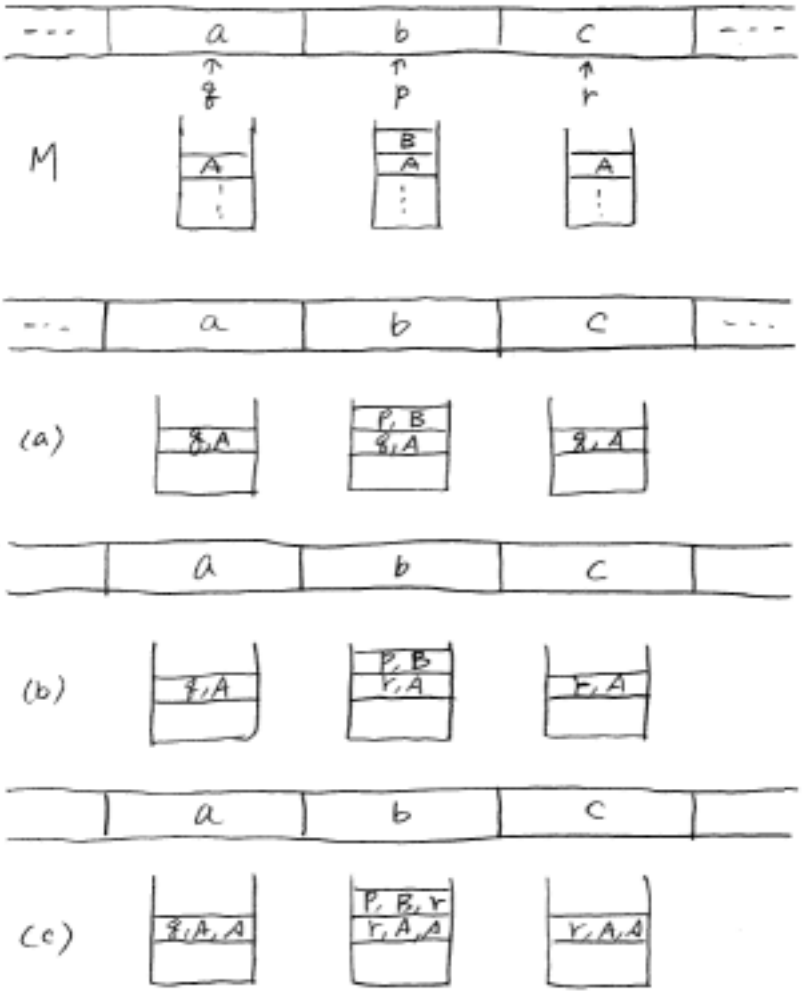

図 4.16 1 状態プッシュダウンオートマトン

1 状態で模倣する基本的アイデアは，例題で述べたように，状態の情報をスタック記号に押し込んでしまうことである．例えば今の場合，a を読んでいる時の状態 q が仮にスタックの先頭記号に A と共に埋め込まれていると仮定しよう（同図 (a)）．すると，1 状態の pda（M_S と呼ぶ）は M の現在の状況（つまり，状態が q でスタックの先頭が A）を M_S のスタックの先頭を見るだけで知ることができる．従って M の動作を完全に模倣することができ，スタックに記号 B をプッシュするときに，M の次の状態を一緒に埋め込んでしまうことができる．この後も M_S は M の動作を完全に模倣でき，スタック記号をポップすることになる．ここで不都合が生じた．つまり，スタックの下から現れた記号に埋め込まれた状態の情報（つまり状態 q）が現実（r）と合致しないのである．

この不都合の原因ははっきりしている．スタック記号 (q, A) の状態の情報 q は今より 2 ステップ以前の状態である．その状態情報が一旦その上に記号が置かれることによって隠れ，その上の記号が除かれることによって再び現れたのであるから本来の状態の変化を追随できるはずがない．

そこでゲスを使うのである．つまり，a を読んで，スタックの上に記号を積み上げる時に，後

刻その記号が取り去られて，再び今の記号がスタックの先頭になったときの M の状態をゲスしてあらかじめ書き換えておくのである．今の場合だったら，(q, A) の上に記号が乗って隠されてしまう時に，再び現れた時の M の状態を r とゲスして，(r, A) に変えておく．（これは先頭の記号を列で置き換えるという定義にかなっている．）そうすれば，同図 (b) に示される様に，M の状態を正しく追っている．

これだけでは不十分であることはもう読者の方は見抜いておられるに違いない．つまり，ゲスがはずれた時の対策ができていないのである．そこでもう一段の工夫を行う．同図 (c) に示すように，スタック記号に更にもう 1 個の情報を追加する．それは，例えば，a を読んでいるときのスタックの先頭の記号は (q, A, s) であるが，この第 3 の記号 s がその工夫である．この記号の役割は，単にその下の記号の第 1 成分が s であることを示しているだけなのである．従って，そのステップで M を模倣するときに以下の様な仕組みになる．(i) 先頭の記号 (q, A, s) が隠されてしまうので，再び現れた時の M の状態をゲスし（今の場合は r），その第 1 成分を r に置き換える．(ii) 記号をプッシュするときに，次の状態 p を第 1 成分，第 2 成分を本来のスタック記号とする他に更に第 3 成分を用意して，それを直下の記号の第 1 成分 r に合わせる．

こうしておくと，ゲスがはずれたことが分かる．今の場合であったら，図の (c) で真ん中の b を読んでいるときに，模倣によればスタック記号をポップする必要があって，そうすると現在の状態の情報として，r が現れるであろうことが，(p, B, r) の第 3 の要素をみることによって分かるのである．M の状態遷移関数を模倣すると次の状態が r になることが分かるので，以前行った状態に関するゲスは正しかったことがこの時点で分かって，安心して先頭の記号をポップできる．不幸にもゲスが間違っていたことが判明したら，その時点で模倣を中断して停止してしまえば良い．以前にも述べた様に，ゲスが当った時には望ましい動作をしているのであれば，ゲスがはずれた時は何も考えないで非受理にしてしまうのが最も安全である．

シミュレーションのアイデアが分かって頂けたであろうか．これ以上，例えば M_S の状態遷移関数の構成法等に踏み込むとかなりややこしくなるので，ここでやめておこう．（証明終り）

定理 4.2　文脈自由文法とプッシュダウンオートマトンは等価である．

（証明）まず，プッシュダウンオートマトンから文脈自由文法を構成する方法を与える．まず与えられた pda を前定理によって 1 状態に直す．その後は，練習問題 4.5 の様に状態遷移関数の書き下しから完全に一体一の生成規則が得られる．すなわち

$$(s, \alpha) \in \delta(s, a, A) \quad と \quad A \to a\alpha$$

を対応させる形で生成規則を導く．逆は，まず与えられた文法を前章の定理 3.5 によってグライバッハの標準形に直す．すると，全ての生成規則は変数のみからなる列 α に対して，$A \to a\alpha$ と書けるので，上の対応をそのまま使って，状態遷移関数を書き下すことができる．この構成法が正しいことは，問題 4.5 の例から明らかであろう．（証明終り）

4.4　まとめと文献

プッシュダウンオートマトンを最初に提唱した人は Oettinger と Schutzenberger であるとホップクロフト・ウルマンの教科書に書いてある．また，それによれば，pda と文脈自由文法の等価性を証明した人は Chomsky と Evey である．これらの仕事は全て 60 年代の初めである．

筆者が学生であった 70 年代の後半でも，文脈自由文法・プッシュダウンオートマトンの研究はかなり活発に行われていた．そのころにバイブルの様に扱われていたのが以下の本で，これは教科書というより完全な専門書である．難解で，私は最初の 10 分の 1 くらいのところで挫折してしまった．

・S. Ginsburg, The mathematical theory of context-free languages, McGraw-Hill, 1966.

この Ginsburg という人には筆者も何回かあったことがある．実は筆者が計算機の理論（最初は典型的なオートマトン理論であった）を本格的に始めたころは， Ginsburg は既に研究分野をデータベースの理論にシフトしていてた．私はデータベースは余り興味を持っていなかったので，本来なら遭遇するチャンスも得られなかったのだと思う．しかし，当時筆者が所属した大学の研究室がデータベースの研究をかなり精力的に進めていたので，その関連でお会いするチャンスに恵まれたのである．始めて米国の学会で発表することになった時に，彼を南カルフォルニア大学に訪ねた．色々アドバイスを頂いたのであるが，今でも良く覚えていることは，理論の会議では絶対にスーツを着るな，というアドバイスであった．私は勿論その時もスーツを着ていったのである，その後学会に行ってみて良く分かった．スーツを着ていたのは， 300 人くらいの参加者の中のほんの数人で，田舎者丸出しという感じで，本当に恥ずかしい思いをした．それからは，私も，日本ではそれなりの服装をしているが，外国へ行くときはラフな服装に徹することにしている．

プッシュダウンオートマトンを勉強した人は様々な類似のオートマトンを思いつくであろう．例えば，補助記憶のスタックをカウンター（つまりスタックにプッシュできる記号を 1 種類に限定する）に置き換えたモデル，入力テープのヘッドを左にも動くことができるようにしたモデル，スタックを 1 個だけでなく 2 個以上使えるようにしたモデル，等々際限がない．実際， 60 年代から 70 年代にかけて，それこそ数百にも達する程数多くのオートマトンモデルが出現した．微妙な補助記憶の違いで能力の違いが現れるというのが典型的な議論のスタイルであった．ある意味でパズルのような楽しさがあって，結構多くの研究者が「はまってしまった」らしい．しかし，これは残念ながら学会の主流にはならなかった．むしろ，モデルの脆弱性（つまり，あまり重要でない細部を無視することができない）をさらけ出したのである．これを良い経験として，頑健なモデルで議論するという基本的仕組みができあがって，現在に至っている．

プッシュダウンオートマトンに関連して触れておかねばならないことがある．それは決定性 pda の等価性判定問題である．まず決定性の説明であるが，これは文字通り，状態遷移関数の右辺の集合の要素が 1 以下ということである（これも色々細部があるのだがここでは触れない）．等価性判定問題とは 2 つの pda が同じ言語を認識するかどうかを判定する問題である．以前に議論したように有限オートマトンの場合は簡単で，直積オートマトンを作ることによって判定できる．実は，非決定性 pda の場合はこの問題は非可解になるのである．（非可解性についてはもうすぐ説明する．ここでは，その問題を解く計算機のプログラムが書けないと思っておいて頂ければよいであろう）．非決定性の場合はこのように等価性判定は非常に難しいのであるが，決定性の場合は解けそうに見える（つまり， 2 つの決定性 pda を入力すると，等価か否かを答えるプログラムが書けそうにみえる）．この問題は 30 年以上に渡って，最も有名な未解決問題の一つであった．色々制限を付ければ解けるという議論が続いて，これには日本人の研究者も大いに貢献している． 90 年代の終りに，とうとう最終的な解決が得られた．以下の論文が最も詳しいらしいが，そのページ数に注目してほしい．

・ G. Sénizergues, “L(A)=L(B)? decidability results from complete formal systems,” Theoretical COmputer Science, 251, 1-166, 2001.

理解度の確認

問　様相が 1 ステップで移れるという定義を形式的に書け．（ヒント： 2 つの様相 C_1 と C_2 は以下の条件を満たすとき 1 ステップで移れると言う．条件： $C_1 = (s, ay, A\alpha)$， $C_1 = (s', y, \beta\alpha)$ と書ける．状態遷移関数が何々の条件を満たす．といった感じでやって欲しい．）

問　練習問題 4.2 の初期様相を根として，全ての様相の変化を表す木を書き下せ．非決定性なので， 1 本の道ではなく分岐のある木になるのである．

問　練習問題 4.3 の pda の状態遷移関数を書き下せ．

問　習問題 4.4 の機械を 1 状態で作れ．また，この言語を生成する文脈自由文法を求めよ．

問　$\{x \mid x \in \{0,1\}^* \text{且つ} x = x^{\text{scriptsize R}}\}$ を認識する pda を設計せよ．

問　$\{x \mid x \in \{0,1\}^* \text{且つ} \sharp_0(x) = \sharp_1(x)\}$ を認識する pda を設計せよ．（ヒント：簡単だとは思うが， x の途中で 0 の個数と 1 の個数が同じになることもあるので注意が必要である．）

問　$\{0^n 1^m \mid n \neq m \text{かつ} n \neq 2m\}$ を認識する pda を設計せよ．（ヒント：先に $\{0^n 1^m \mid n < m < 2m\}$ を認識する pda を考えて見てはいかがであろうか．）

問　練習問題 4.4 の解答の最後で，「有限状態を利用して記憶し」と書いたが，具体的にどのようにすれば良いのかを詳しく述べよ．（ヒント： fa に以下の様な補助記憶を付加したモデルを考えよう．補助記憶はレジスタと呼ばれ， R_i で表す． 1 個のレジスタ当り， 0 又は 1 の 1 ビットが記憶できる． 1 ステップ動作は，現在の状態だけでなく，レジスタの値にも依存してよい．レジスタの個数が有限ならそれは普通の fa でシミュレートできる． 1 個のレジスタごとに状態数を 2 倍に増やせば良い．）

問　我々の pda はスタックを空にすることによって受理するが， fa の場合と同じ様に，受理状態に到達して受理するようなモデルも考えることができる．そのようなモデルを定義せよ．勿論，今のモデルと等価にならなければ意味が無い．（ヒント：今のモデル（空スタック式と呼ぶ）にちょっとして工夫を加えることによって，今読んでいるスタック記号がスタックの一番下の記号，つまりそれをポップするとスタックは空になってしまう，かどうかが分かる様にすることができよう．その後で，空スタック式の pda M をシミュレートする．現在読んでいるテープの記号が最後の記号であるかどうかをゲスする機構を加える．最後だとゲスして，且つスタックの先頭も一番下の記号で，更に M がスタックの記号をポップするときのみ受理状態に遷移する．受理状態では完全に停止する．こうすれば受理条件は，「テープを読み終った時に受理状態にいる」ということにならないであろうか．）このようなモデルの利点（ある種の言語に関しては設計が簡単になる）に関しても考察せよ．例えば，練習問題 4.3 で考えた言語は全く異なった方法で処理できるのではないか．

問　更に異なったモデルとして ϵ 遷移を許すモデルを考えよう．これは， fa の時と同じように，入力ヘッドを動かさずに 1 ステップ動作を行うことを許すモデルである． 1 ステップ動作は今の

モデルと同じである．つまり，状態遷移関数は

$$\delta(s_1, \epsilon, A) = \{(s_2, \epsilon), (s_3, AA)\}$$

の様になる．このモデルは幾つかの特徴がある．

（1）スタック動作は，本来のプッシュ，ポップ，先頭の記号を読む，という 3 種類のみで十分であることを示せ．

（2）文脈自由文法をシミュレートするとき，今の pda ではグライバッハの標準形に直す必要があった．ϵ 遷移を許すと，このような制限は取り外せて，ほとんどどんな文法（例えば，チョムスキーの標準形）でもシミュレートできることを示せ．（ヒント：シミュレーションを復習すると，規則 $A \to aBB$ の右辺の a がテープで読む記号に対応している．右辺が変数で始まっているなら，テープの記号を読まないでスタック操作を行えば良い．）

（3）以上はモデルの強力化によって得られる利点と捉えることができるかもしれないが，当然複雑になるデメリットも存在する．例えば，ϵ 遷移があると，入力の受理・非受理を決定するのに要するステップ数が延びる可能性がある．今の pda はテープの長さが n であるなら必ず n ステップ以内の動作で停止するが，ϵ 遷移を許すなら，永遠に停止しないことすら考えられる．そのような「停止しない」状況を回避できるか．

（4）ϵ 遷移を許す pda は今の pda より真に強力であろうか．

問　決定性の pda（状態遷移関数の右辺の要素の数が 1 以下）で更に以下の様な条件を満たすものを考える．(i) スタック記号は 1 種類（すなわちスタック初期記号）しか使えない．(ii) 初期様相からスタートして，スタックの中の文字数が増加して行くが，一旦減少したら再び増加することはない．(iii) 入力記号は 0 と 1 のみとする．このような条件を満たす，2 つの決定性 pda が与えられた時，それらが同じ言語を認識するかどうか判定する方法を考えよ．（ヒント：fa の場合と同じ様に，直積機械を作る．但し，今度はスタックに保持できる記号数が有限では無いので，単純に作れば直積機械は有限状態では納まらない．何とかして，有限状態に納める努力を行って欲しい．）