オートマトンと計算理論

第１部
正規表現と有限オートマトン

火曜５・６限目　必修科目

尾張　正樹

居室：J2415 (情報２号館４階)

masakiowari@inf.shizuoka.ac.jp

講義資料：¥¥fs.inf.in.shizuoka.ac.jp¥share¥class¥2019オートマトン

本講義の評価について

- 本講義の成績評価は、小テスト、期末試験、誤植の指摘と再試験により行います。

- 小テスト (予告せずに複数回実施)を40点満点、期末試験を60点満点で採点を行います。

- 上記合計100点に『誤植の指摘』によるおまけの点数を加えた点が60点以上で合格とします。

再試験について

『小テスト+期末試験+誤植の指摘』が60点に満たなかったが、下記の条件①②を共に満たしている学生には、再試験を受ける機会を与えます：

①『小テスト』が20点以上

②『小テスト+期末試験+誤植の指摘』が30点以上

- 再試験は60点満点で採点をし、『小テスト+再試験+誤植の指摘』が80点以上取った場合にのみ単位を与えます。
- 再試験は2月下旬を予定しています。

『小テスト+期末試験+誤植の指摘』による評価の結果は、期末試験終了後、比較的早期にお知らせする予定です。

再試験に付随する規則

再試験は、静岡大学の規則では行う義務が存在せず、尾張の善意で行っているものです。

このことを鑑みて以下の規則を適応します：

① どのような理由があっても、再試験の追試は行わない。

② どのような理由があっても、再試験対象者の開示も含めて、期末試験の成績の開示を特定の学生のみに早く行うことはしない。

③ 再試験の時期については、教室の空き状況や、尾張自身や学科・学部のスケジュール変更などに伴い、再試験を行う時期が予告とは異なる可能性がある。その場合でも、上記①②の規則は適応される。

誤植の指摘によるおまけの点数

- 講義資料や期末試験候補問題集には、私の不注意から誤植(書き間違い)が含まれている可能性があります。
- 講義中に私が気づけなかった誤植を後日メールなどで最初に指摘した学生には、誤植の程度(重要さ)により０～５点を与えます。
- 誤植の指摘による点数は『課題提出＋期末試験』の点数に加算します。
- 『てにおは』の間違いや『漢字の書き間違い』などの数学的内容に関係のない間違いは指摘をしても点数にはなりません（0点加算）。
- 数学的内容が異なってしまうような誤植（添え字や変数が間違っているなど）の場合は、１～５点の点数を加算します。
- １箇所の誤植につき点数をもらえる学生は１人のみです。早い者勝ちです。
- 誤植の指摘は、メールなどの証拠が残る方法で行い、名前と学籍番号を必ず書いてください。

小テストについて

- 講義中に不定期に予告なく実施
- 3～5回実施
- 制限時間は約10～20分 (毎回異なる)
- 学務情報システム(Web小テスト)を使って解答
- 参考書やノートは見てはいけない
- PCで講義資料(PDF)は見てもよいが、学務情報システム以外のインターネットへの接続は禁止。

- 期末試験候補問題より簡単な問題が出題される

期末試験および再試験について(1)

- 制限時間は約80分です。
- 参考書やノート、PCなどを試験中に見てはいけません。
- 問題は3～6題ほど出題されます。
- 問題は、期末試験候補問題集より出題します。
- 問題中の一部のみが出題されたり、複数の問題が1題として出題されたりすることもあります。

期末試験および再試験について(2)

期末試験による成績の評価は、下記の要領で行います。

・問題N題中、M題正解すれば、$60 \times \frac{M}{N}$点とする。

・小テスト（40点満点）を加算して評価とする。

再試験による成績の評価は、下記の要領で行います。

・問題N題中、M題正解すれば、$60 \times \frac{M}{N}$点とする。

・『小テスト+再試験+誤植の指摘』が80点以上取った場合にのみ単位(60点)を与えます。

参考書

メイン:『オートマトン・言語と計算理論』
岩間一雄 (コロナ社)

サブ参考書:上記で足りないところの補足用

- 『計算理論の基礎』(第1、2巻)
  Michael Sipser (共立出版)
- 『アルゴリズムと計算量』
  谷聖一 (サイエンス社・SGCライブラリ43)
- 『計算理論とオートマトン言語理論』
  丸山章 (サイエンス社)
- 『オートマトン・言語理論』
  富田悦次・横森貴 (森北出版)　　・・・など。

講義予定

イントロダクション(1回目)

第1部　正規表現と有限オートマトン

1章. 形式言語の導入 (1-2回目)

2章. 正規表現と有限オートマトン (2-5回目)

第2部　文脈自由文法とプッシュダウンオートマトン

1章. 文脈自由文法 (5-7回目)

2章. プッシュダウンオートマトン (7-9回目)

第3部チューリング機械と計算理論

1章. チューリング機械と0型文法 (9-12回目)

2章. チューリング機械の停止性と決定問題 (12-13回目)

3章. NP完全問題 (14-15回目)

第0部：イントロダクション

この講義で何を学ぶのか？

計算とは何か？(1)

情報科学科で学んで欲しいこと:

『計算機とは何か？』

Computer＝計算機

この講義の目的:

『計算機とは何か？』を数学的に理解する。

計算とは何か？(2)

『計算機とは何か？』

Computer＝ 計算機

＝ 計算(computing)する機械

(注意) 我々のComputerは『通信』も行っているが、
この講義では計算のみを扱う。
(⇔ネットつながっていないPC)

計算とは何か？(3)

計算(computing)って何？

- 四則演算とその組み合わせ:

$$10 + 5 = 15,\ 3 - 9 = -6,$$

$$12 \times 10 = 120, \qquad 16 \div 3 = \frac{16}{3} = 5.333..$$

$$(((2 + 1) \times 4) + (4 \div 2)) \times 3 = 42,$$

- 微分積分 : $\frac{d(2x^2+4x-1)}{dx}|_{x=1} = 8$

$$\int_0^1 2x^2 + 4x - 1\, dx = \frac{5}{3} = 1.666..$$

線形代数や離散数学の計算も皆さんは知ってます。

計算とは何か？(4)

計算(computing)って何？

- 四則演算とその組み合わせ:

$10 + 5 = 15, \quad f_+ : (x_1, x_2) \mapsto x_1 + x_2$

$3 - 9 = -6, \quad f_- : (x_1, x_2) \mapsto x_1 - x_2$

$12 \times 10 = 120, \quad f_\times : (x_1, x_2) \mapsto x_1 \times x_2$

$16 \div 3 = \frac{16}{3} = 5.333.. \quad f_\div : (x_1, x_2) \mapsto x_1 \div x_2$

$$(((2 + 1) \times 4) + (4 \div 2)) \times 3 = 42,$$

$$g(x_1, x_2, x_3, x_4) := f_\times\big(f_+\big(f_\times(f_+(x_1, x_2), x_3), f_\div(x_3, x_1)\big), x_4\big)$$

計算とは何か？(5)

計算(computing)って何？

- 微分積分：

$$\left.\frac{d(2x^2+4x-1)}{dx}\right|_{x=1} = 8$$

① $g(x,a,b,c) = ax^2 + bx + c$に対して、
導関数を計算：$g'(x,a,b) = 2ax + b$
$= f_+\big(f_\times\big(2, f_\times(a,x)\big), b\big).$

② $\left.\frac{d(2x^2+4x-1)}{dx}\right|_{x=1} = f_+\big(f_\times\big(2, f_\times(2,1)\big), 4\big) = 8$

計算とは何か？(6)

計算(computing)って何？

- 微分積分：$\int_0^1 2x^2 + 4x - 1\, dx = \frac{5}{3} = 1.666..$

① $g(x, a, b, c) = ax^2 + bx + c$に対して、
原始関数を計算：

$$G(x, a, b, c) = \frac{a}{3}x^3 + \frac{b}{2}x^2 + cx$$
$$= f_+(\ f_+(\ f_\times\left(f_\div(a, 3), f_\times\left(f_\times(x, x), x\right)\right),$$
$$f_\times\left(f_\div(b, 2), f_\times(x, x)\right)\ ), f_\times(c, x)\ )$$

計算とは何か？(7)

計算(computing)って何？

- 微分積分 : $\int_0^1 2x^2 + 4x - 1dx = \frac{5}{3} = 1.666..$

② $\int_0^1 2x^2 + 4x - 1\, dx$

$= G(1,2,4,-1) - G(0,2,4,-1)$

$= f_+(\ f_+(\ f_\times\big(f_\div(2,3), f_\times(f_\times(1,1),1)\big),$
$f_\times\big(f_\div(4,2), f_\times(1,1)\big)\ ), f_\times(-1,1)\ )$
$-f_+(\ f_+(\ f_\times\big(f_\div(2,3), f_\times(f_\times(0,0),0)\big),$
$f_\times\big(f_\div(4,2), f_\times(0,0)\big)\ ), f_\times(-1,0)\ )$

$= \frac{5}{3} - 0 = \frac{5}{3} = 1.666\ldots$

何を計算するのか？(1)

計算(computing)って何？

⇒そもそも『何を』計算するのか？

与えられた入力に対して、
関数$f(x_1, x_2, \cdots, x_n)$の値を『計算』する。

先ほどの例：

① $f(x_1, x_2, \cdots, x_n)$を四則演算$f_+, f_-, f_\times, f_\div$で表す。

$$f(x_1, x_2, x_3, x_4) = f_\times\left(f_+\left(f_\times(f_+(x_1, x_2), x_3), f_\div(x_3, x_1)\right), x_4\right)$$

① $f_+, f_-, f_\times, f_\div$の計算の仕方は知っているので、
$f(x_1, x_2, \cdots, x_n)$の値が計算できる。

何を計算するのか？(2)

先ほどの例：

① $f(x_1, x_2, \cdots, x_n)$を四則演算$f_+, f_-, f_\times, f_\div$で表す。

② $f_+, f_-, f_\times, f_\div$の計算の仕方は知っているので、
$f(x_1, x_2, \cdots, x_n)$の値が計算できる。

https://commons.wikimedia.org/wiki/File:Ono_City_Tradition_Industrial_Hall05s4272.jpg

②は昔は『そろばん』でやっていた。(最古の『計算機』)

何を計算するのか？(3)

与えられた入力に対して、
関数$f(x_1, x_2, \cdots, x_n)$の値を『計算』する。

（論理回路や数理論理より）
通常のPCの中では、ブール関数$f: \{0,1\}^n \to \{0,1\}$
の値を論理回路を用いて計算している。

例：$f(x_1, x_2, x_3, x_4) = \big((x_1 \wedge x_2) \vee \neg x_3\big) \supset (x_1 \vee \neg x_4)$
$= f_{\supset}\Big(f_{\vee}\big(f_{\wedge}(x_1, x_2), f_{\neg}(x_3)\big), f_{\vee}\big(x_1, f_{\neg}(x_4)\big)\Big)$

ここで、$x_1, x_2, x_3, x_4 \in \{0,1\}$,
$f_{\vee}(x, y) \coloneqq x \vee y, f_{\neg}(x) \coloneqq \neg x$・・・・など

何を計算するのか？(4)

通常のPCの中では、ブール関数$f:\{0,1\}^n \to \{0,1\}$
の値を論理回路を用いて計算している。

例: $f(x_1,x_2,x_3,x_4) = \big((x_1 \wedge x_2) \vee \neg x_3\big) \supset (x_1 \vee \neg x_4)$
$= f_{\supset}\Big(f_{\vee}\big(f_{\wedge}(x_1,x_2), f_{\neg}(x_3)\big), f_{\vee}\big(x_1, f_{\neg}(x_4)\big)\Big)$

① $f(x_1,x_2,\cdots,x_n)$を論理演算$f_{\vee}, f_{\wedge}, f_{\neg}, f_{\supset}$で表す。
論理回路を描く(プログラムを書く)のと同じ

② $f_{\vee}, f_{\wedge}, f_{\neg}, f_{\supset}$の計算をする論理素子が実装されているので$f(x_1,x_2,\cdots,x_n)$の値が計算できる。

何を計算するのか？(5)

複数ビットの出力を持つ関数$f:\{0,1\}^n \to \{0,1\}^k$

k個のブール関数$f_1, \cdots, f_i, \cdots, f_k$

$f_i:\{0,1\}^n \to \{0,1\}$を用いて、

$$f(x_1, \cdots, x_n) = \left(f_1(x_1, \cdots, x_n), \cdots, f_k(x_1, \cdots, x_n)\right)$$

1ビット出力のブール関数$f_1, \cdots, f_i, \cdots, f_k$が計算できれば、$f$は計算できる。

何を計算するのか？(6)

最初の例：　　$f: \mathbb{N} \times \cdots \times \mathbb{N} \to \mathbb{N}$

論理回路の例：　$f: \{0,1\}^n \to \{0,1\}^k$

自然数$\mathbb{N}$を2進数表示すると、

$$1 \to 00 \cdots 001$$
$$2 \to 00 \cdots 010$$
$$3 \to 00 \cdots 011$$
$$\vdots$$
$$n \to x_1 x_2 \cdots x_m$$

と、自然数をビット列で表現できた。

（実際は、$m$bitでは、$0 \sim 2^m - 1$までしか表現できない。）

何を計算するのか？(7)

自然数をビット列で表現できた。

実際は、$n$bitでは、$0 \sim 2^n - 1$までしか表現できない。

もっとも、大きな自然数を計算するには、
非常に大きな『そろばん』が必要　⇔　メモリの有限性

メモリの制限がなければ、いくらでも大きな論理回路を実装可能なはず。

全てのビット列の集合を以下で定義：

$$\{0,1\}^* := \bigcup_{n \in \mathbb{N}} \{0,1\}^n$$

メモリの制限のない(架空の)PCは、$f : \{0,1\}^* \to \{0,1\}^*$を計算

何を計算するのか？(8)

全てのビット列の集合を以下で定義：

$$\{0,1\}^* := \bigcup_{n\in\mathbb{N}} \{0,1\}^n$$

メモリの制限のない(架空の)PCは、$f:\{0,1\}^* \to \{0,1\}^*$を計算

離散数学で、
『可算集合の可算個の和集合(=可算集合× 可算集合)は、可算集合』
であることを習った。

$\{0,1\}^*$と$\mathbb{N}$は対等(⇔全単射 $g:\{0,1\}^* \to \mathbb{N}$ が存在)。

何を計算するのか？(9)

$\{0,1\}^*$と$\mathbb{N}$は対等(⇔全単射 $g:\{0,1\}^* \to \mathbb{N}$ が存在)。

関数$f:\mathbb{N} \to \mathbb{N}$は、
ビット列の写像$g^{-1} \circ f \circ g:\{0,1\}^* \to \{0,1\}^*$ に変換できる。

ビット列の写像$f:\{0,1\}^* \to \{0,1\}^*$
関数$g \circ f \circ g^{-1}:\mathbb{N} \to \mathbb{N}$に変換できる。

（注意）実際は、gは全単射でなくても、単射であればよく、
その場合、g^{-1}の代わりに、別の単射$g':\mathbb{N} \to \{0,1\}^*$を使う。

何を計算するのか？(10)

そもそも、通常、自然数$n \in \mathbb{N}$は、10進法で扱っている。

1,2,3,...9,
10,11,12,....99,
100,101....,999,

10進法では、自然数nに$\{0,1,\cdots,9\}$の列$x_1 x_2 \cdots x_m$を対応させる

全ての$\{0,1,\cdots,9\}$の列の集合を以下で定義:

$$\{0,1,\cdots,9\}^* := \bigcup_{n\in\mathbb{N}} \{0,1,\cdots,9\}^n$$

$\{0,1,\cdots,9\}^*$と$\mathbb{N}$も対等

$\Leftrightarrow$全単射 $g:\{0,1,\cdots,9\}^* \to \mathbb{N}$ が存在。

何を計算するのか？(11)

$\{0,1,\cdots,9\}^*$とℕも対等

⇔全単射 $g:\{0,1,\cdots,9\}^* \to \mathbb{N}$ が存在。

関数$f:\mathbb{N} \to \mathbb{N}$は、$\{0,1,\cdots,9\}$の列の写像

$g^{-1} \circ f \circ g:\{0,1,\cdots,9\}^* \to \{0,1,\cdots,9\}^*$ に変換できる。

$\{0,1,\cdots,9\}$の列の写像$f:\{0,1,\cdots,9\}^* \to \{0,1,\cdots,9\}^*$

関数$g \circ f \circ g^{-1}:\mathbb{N} \to \mathbb{N}$に変換できる。

何を計算するのか？(12)

任意の自然数mに対してm進法が存在する。$(m \geq 2)$

同様の議論により、

関数$f: \mathbb{N} \to \mathbb{N}$は、$\{0,1,\cdots,m-1\}$の列の写像
$f': \{0,1,\cdots,m-1\}^* \to \{0,1,\cdots,m-1\}^*$ に変換できる。

$$\{0,1,\cdots,m-1\}^* := \bigcup_{n\in\mathbb{N}} \{0,1,\cdots,m-1\}^n$$

何を計算するのか？(13)

そもそも、アラビア数字を使う必要性はない。

任意の記号(文字)の集合(アルファベット)$\Sigma := \{a_1, \cdots, a_m\}$は、

$\{0,1, \cdots, m-1\}$と対等。

$\Sigma := \{a_1, \cdots, a_m\}$の列を以下で定義：

$$\Sigma^* = \{a_1, \cdots, a_m\}^* := \bigcup_{n \in \mathbb{N}} \{a_1, \cdots, a_m\}^n$$

Σ^*と$\mathbb{N}$は対等なので、

関数$f: \mathbb{N} \to \mathbb{N}$は、$\Sigma$の列の写像$f': \Sigma^* \to \Sigma^*$ に変換できる。

何を計算するのか？(14)

計算(computing)って何？
⇒そもそも『何を』計算するのか？

答え：記号の列Σ^*の写像$f: \Sigma^* \to \Sigma^*$

$\Sigma = \{0,1\}$のときはビット列の写像になる。

Σ^*と$\mathbb{N}$は対等な集合なので、
自然数の関数$f: \mathbb{N} \to \mathbb{N}$を考えているのと同じである。

どのように計算するのか？(1)

計算(computing)って何？
⇒『どのように』計算するのか？

通常、人間(手計算、そろばん)やPC(機械)は、
定式化された手順（アルゴリズム）に従って計算を進めている。

アルゴリズムは、人間や機械が、
精確に順を追って従うことのできる規則のリストとして指定される。

人間が計算を行う動作を、単純化していくことで、
『計算の仕方』を数学的に定義する。

どのように計算するのか？ (2)

人間(Aさん)が計算を行う動作を、単純化していくことで、
『計算の仕方』を数学的に定義する。

Aさんは、紙とペンで計算を進めていくとする。

4231
×77
29617
296170
325787

左図のように、掛け算を計算する。

しかし、この計算は一列に書いても、
本質は失わない。

$$4231 \times 77 = 29617 + 296170 = 325787$$

どのように計算するのか？ (3)

$$4231 \times 77 = 29617 + 296170 = 325787$$

Aさんは、マス目が書いてある一次元の非常に長いテープを使って、上記の計算を行うことにする。

4	2	3	1	×	7	7	=	2	9	6	1	7	+	2	9	6	1	7	0	=	3	2	5	7	8	7		

Aさんは、

- テープに沿って行ったり来たりしながら、
- 記号を書きくわえたり、
- 書き込まれた記号を消したり

して計算を進めていく。

どのように計算するのか？ (4)

- テープに沿って行ったり来たりしながら、
- 記号を書きくわえたり、
- 書き込まれた記号を消したり

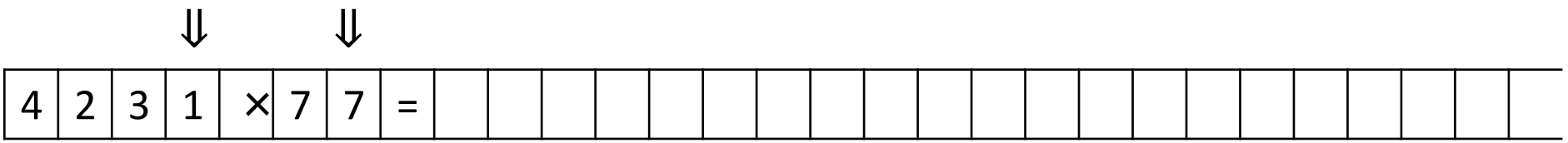
Aさんが、『次にどの記号を書き込むか』は、
『Aさんが注意を向けている記号』 と
『Aさんの心の状態(もしくは脳の状態)』で決まる。

4	2	3	1	×	7	7	=	2	9	6	1	7	+	2	9	6	1	7	0	=	3	2	5	7	8	7		

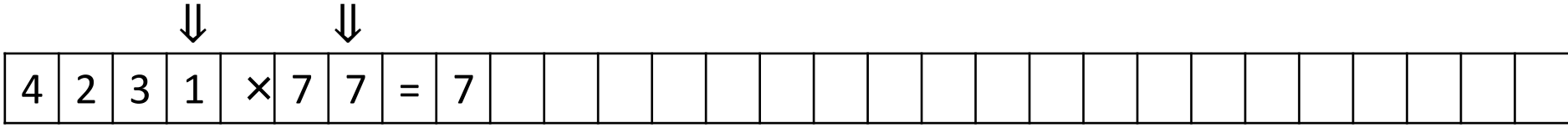
どのように計算するのか？ (5)

Aさんが、『次にどの記号を書き込むか』は、
『Aさんが注意を向けている記号』 と
『Aさんの心の状態(もしくは脳の状態)』で決まる。

ここにAさんの注意が向けられているとする。

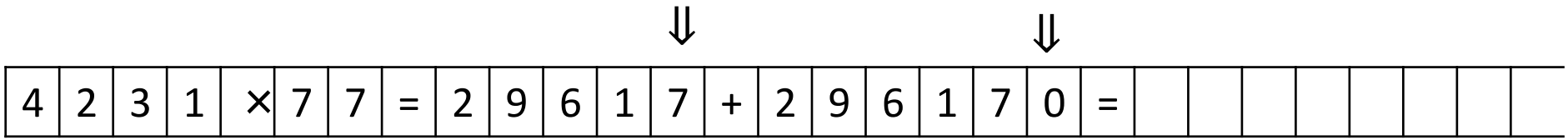


1かける7は7なので、Aさんは7を書き込む

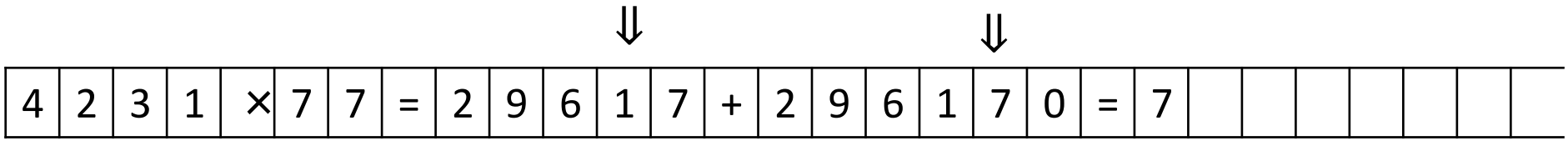


どのように計算するのか？(6)

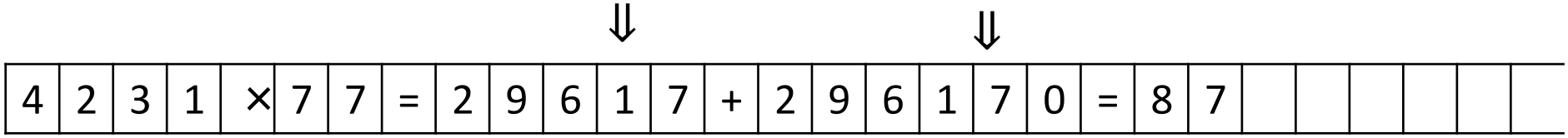
ここにAさんの注意が向けられているとする。



7たす0は7なので、7を書き込む。注意を動かす。



7たす1は8なので、8を書き込む（先ほどは掛け算だった！）



どのように計算するのか？(7)

Aさんが掛け算をするのか、足し算をするのかは、
Aさんの心の状態に依存する。

一般に、計算を遂行する人物(機械)は、
以下の制約の下で作業を行う:

① 計算過程の各段階において、少数の記号だけに注意が向けられる

② 各段階の動作の決定は、注意を向けられている記号と、計算を実行している人物の現在の心の状態だけに依存する。

どのように計算するのか？ (8)

① 計算過程の各段階において、少数の記号だけに注意が向けられる

どれだけの数の記号に同時に注意を向ける必要があるか？

同時に複数の記号に注意を向ける代わりに、

『一つずつ順番に』それらの記号に注意を向けてもよい。

また、注意を向ける位置をテープのあるマス目から遠くの別のマス目に移動させることは、

マス目を一つずつ、

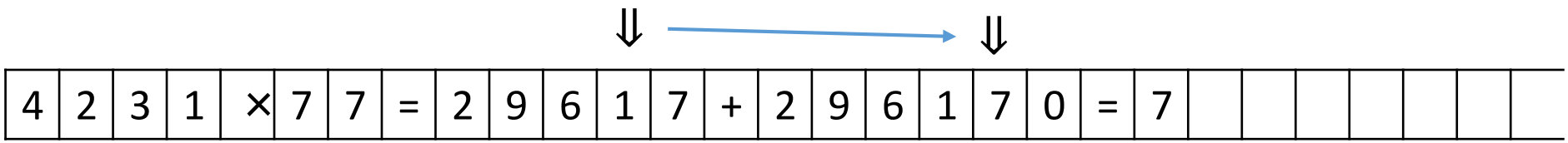
どのように計算するのか？ (9)

① 計算過程の各段階において、少数の記号だけに注意が向けられる

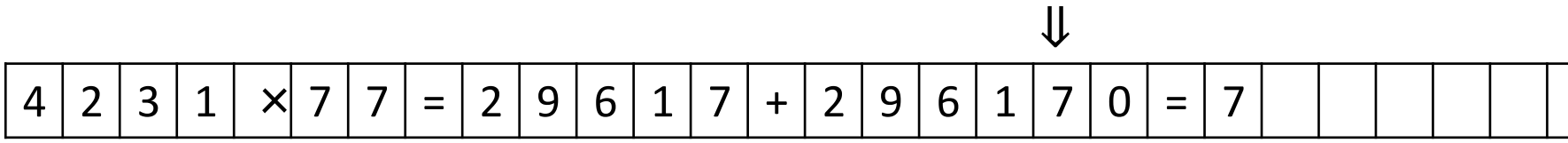
注意を向ける位置の移動について:

注意を向ける位置をテープのあるマス目から遠くの別のマス目に一気に移動させる代わりに、

マス目を一つずつ移動させても本質的に同じ。



どのように計算するのか？(10)



結局、どのような計算作業も以下の手続きで実行できる:

- 計算は、マス目で区切られた紙テープに記号を書くことで遂行される
- 計算過程の各ステップで、ある1つのマス目に書かれた記号に注意を向ける
- 各ステップでの動作:

  ① 注意を向けていたマス目に記号を書き込む(上書きする)、

  ② 注意を向ける位置を『右か左』に1マス移動させる。
- この動作(書き込む記号と、右か左)は、注意が向けられた記号と、Aさんの心の状態で決定される。

どのように計算するのか？(11)

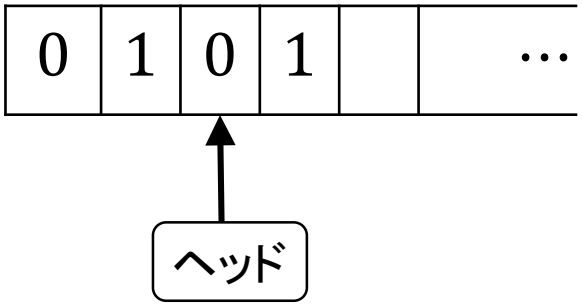
⇓

4	2	3	1	×	7	7	=	2	9	6	1	7	+	2	9	6	1	7	0	=	7						

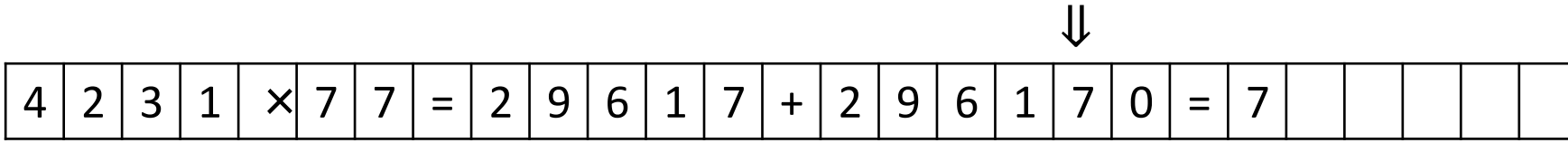
Aさんを機械で置き換える。

Aさんが注意を向ける記号　⇔ 情報を読み書きするヘッド

Aさんの心の状態　　⇔ 機械の内部状態



どのように計算するのか？(12)



機械(チューリング機械)は以下のように動作する:

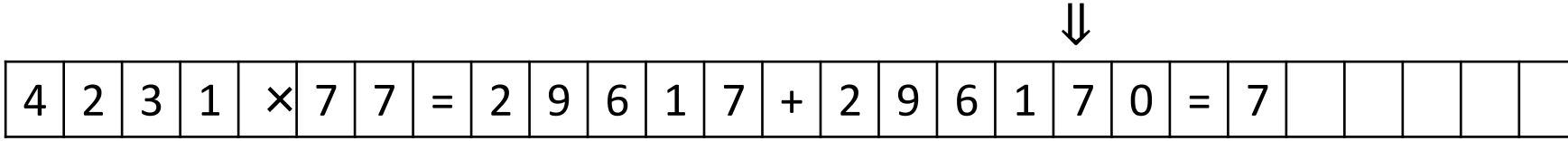
- 機械は、マス目で区切られた紙テープにヘッドを使って記号を読み書きして、計算を進める。
- 計算過程の各ステップで、ヘッドはある1つのマス目にある。
- 各ステップでの動作:

  ① ヘッドのあるマス目に記号を書き込む(上書きする)、

  ② ヘッド位置を『右or左』に1マス移動させる。
- この動作(書き込む記号と、右or左)は、ヘッドの位置にある記号と、機械の内部状態で決定される。

写像 δ:(ヘッド位置の記号,内部状態)↦ (書き込む記号、右or左)
が実質的に『アルゴリズム』になっている。

どのように計算するのか？(13)



チャーチ＝チューリングのテーゼ(Church-Turing thesis):

現実の計算機で計算できる関数は、

チューリング機械で計算できる関数である。

逆に、チューリング機械で計算できる関数は、

メモリが十分にあれば、現実の計算機で計算できる。

どのような新しい仕組みの計算機を設計しても、その計算機で計算できる関数は、チューリング機械で計算可能な関数になるという経験則もしくは予測。

どのように計算するのか？(14)

⇓

4	2	3	1	×	7	7	=	2	9	6	1	7	+	2	9	6	1	7	0	=	7					

チューリング機械 ＝ 通常の計算機の数学的モデル(計算モデル)

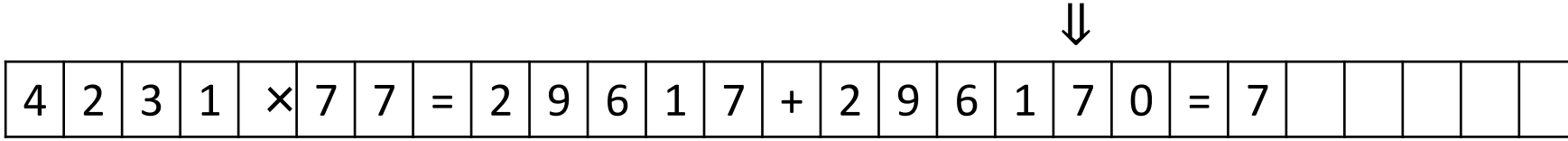
一般に、チューリング機械のように、

『次の動作』が、『現在読んでいる記号』と『内部状態』に依存して決定される計算モデルをオートマトンと呼ぶ。

通常の計算機より『低い能力』しか持たない、

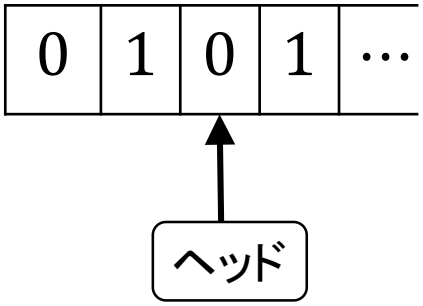
より『簡単な仕組』みの計算機のモデルにも興味がある。

どのように計算するのか？(15)



チューリング機械より『低い能力』しか持たない、

より『簡単な仕組』みのオートマトン：

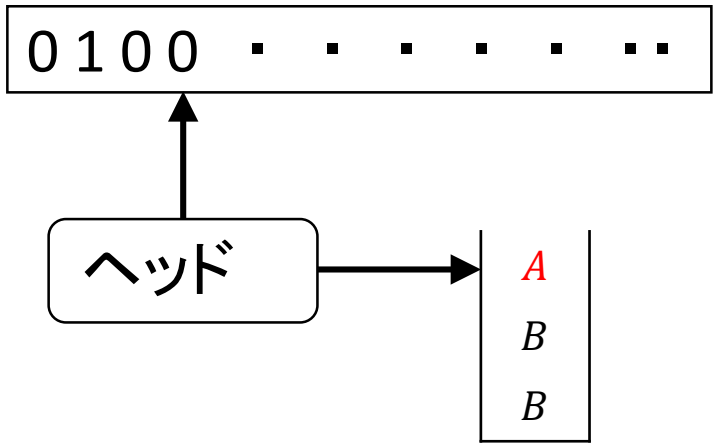


有限オートマトン：テープは読むことしかできない

プッシュダウンオートマトン：テープは読むことしかできない

スタックを持つ

＝情報の出し入れが
先頭からしかできないメモリ)



オートマトンと形式言語 (1)

複数ビットの出力を持つ関数$f: \{0,1\}^n \to \{0,1\}^k$

k個のブール関数$f_1, \cdots, f_i, \cdots, f_k$

$f_i: \{0,1\}^n \to \{0,1\}$を用いて、

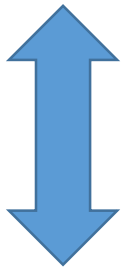
$$f(x_1, \cdots, x_n) = \left(f_1(x_1, \cdots, x_n), \cdots, f_k(x_1, \cdots, x_n)\right)$$

1ビット出力のブール関数$f_1, \cdots, f_i, \cdots, f_k$が計算できれば、$f$は計算できる。

よって、この講義では、当面の間、
1ビットの出力を持つ関数の計算のみを扱う。

オートマトンと形式言語(2)

この講義では、当面の間、
1ビットの出力を持つ関数の計算のみを扱う。



1ビットの出力しか持たないオートマトンのみを扱う。

入力の記号列$x \in \Sigma^*$に対して
オートマトンが1を出力するとき、xを受理する
オートマトンが0を出力するとき、xを拒否する
と呼ぶ。(オートマトンは永遠に止まらない可能性もある)

オートマトンと形式言語(3)

1ビットの出力の関数　$f: \Sigma^* \to \{0,1\}$　は、
$f(x) = 1$となる列の集合 $L := \{x \in \Sigma^* \mid f(x) = 1\}$
を指定すると完全に決定できる。

☆ fは部分集合Lの定義関数である。

fがあるオートマトンMで計算可能

Mは$x \in L$を受理、$x \notin L$を拒否する。

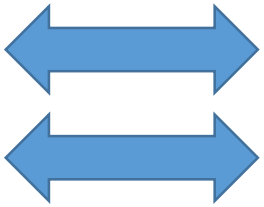
オートマトンと形式言語(4)

記号集合(アルファベット) Σを固定したとき、

記号の列全体の集合 Σ^*の部分集合を言語と呼ぶ。

Lが言語のとき、記号列$x \in L$をLに属する文章と呼ぶ。

オートマトンMで言語Lが認識可能



Mは$x \in L$を受理、$x \notin L$を拒否する。

Lの定義関数fがあるオートマトンMで計算可能

オートマトンと形式言語(4)

オートマトンMで言語Lが認識可能

Mは$x \in L$を受理、$x \notin L$を拒否する。

オートマトンのクラス (チューリング機械、有限オートマトン、・・)
を固定する。
このクラスに属するあるオートマトンMで認識可能な言語を、
このクラスのオートマトンで認識可能な言語と呼ぶ。

チューリング機械で認識可能な言語の集合、
プッシュダウンオートマトンで認識可能な言語の集合、
有限オートマトンで認識可能な言語の集合、　　などを知りたい

オートマトンと形式言語(5)

与えられた言語Lを、コンパクトに（簡潔に）指定したい。

与えられた文章が日本語(英語)であるかどうかは、
文章が日本語(英語)の文法に従って作られているかどうかで、
判断できる。

同様に、言語Lに対して、その文法に従って文章xが作られているかどうかで、Lに入っているかどうかを判断できるもの (形式文法、生成文法)を定義することができる。

“形式言語理論”

オートマトンと形式言語(6)

同様に、言語Lに対して、文章xがLに入っているかどうかを判断する文法(形式文法、生成文法)を定義することができる。

文法のクラスとオートマトンのクラスの対応：

チューリング機械で認識可能な言語の集合、

＝ 0型文法で生成可能な言語の集合

プッシュダウンオートマトンで認識可能な言語の集合、

＝ 文脈自由文法で生成可能な言語の集合

有限オートマトンで認識可能な言語の集合

＝ 正規文法（正規表現）で生成可能な言語の集合

講義全体の前3分の2
＝オートマトンと言語理論

主題は2つ

オートマトン：計算機のモデル(抽象機械)

- 有限オートマトン
- プッシュダウンオートマトン
- チューリング機械

形式言語: (プログラミング)言語のモデル

- 正規表現 ⇔正規言語⇔正規文法
- 文脈自由文法⇔文脈自由言語
- 0型文法⇔0型言語

言語のモデルと計算機のモデルの対応を理解する。

講義全体の後ろ3分の1
＝計算理論

計算模型やアルゴリズムを理論的に扱う分野

計算可能性理論

与えられた問題がコンピュータで解けるか？

⇒チューリング機械の停止性問題

計算複雑性理論

時間計算量：計算にかかるステップ数

空間計算量：計算に必要なメモリ量

特に計算量クラスNPについて詳しく抗議する。

第1部 正規表現と有限オートマトン

第1章：形式言語の導入

1. 文章の意味を理解する
2. 文章の正しさを理解する
3. 形式言語を導入する

1.文章の意味を理解する(1)

計算機に文章の意味を理解させるにはどうしたらよいか？

数式を文章の例にとって考える:

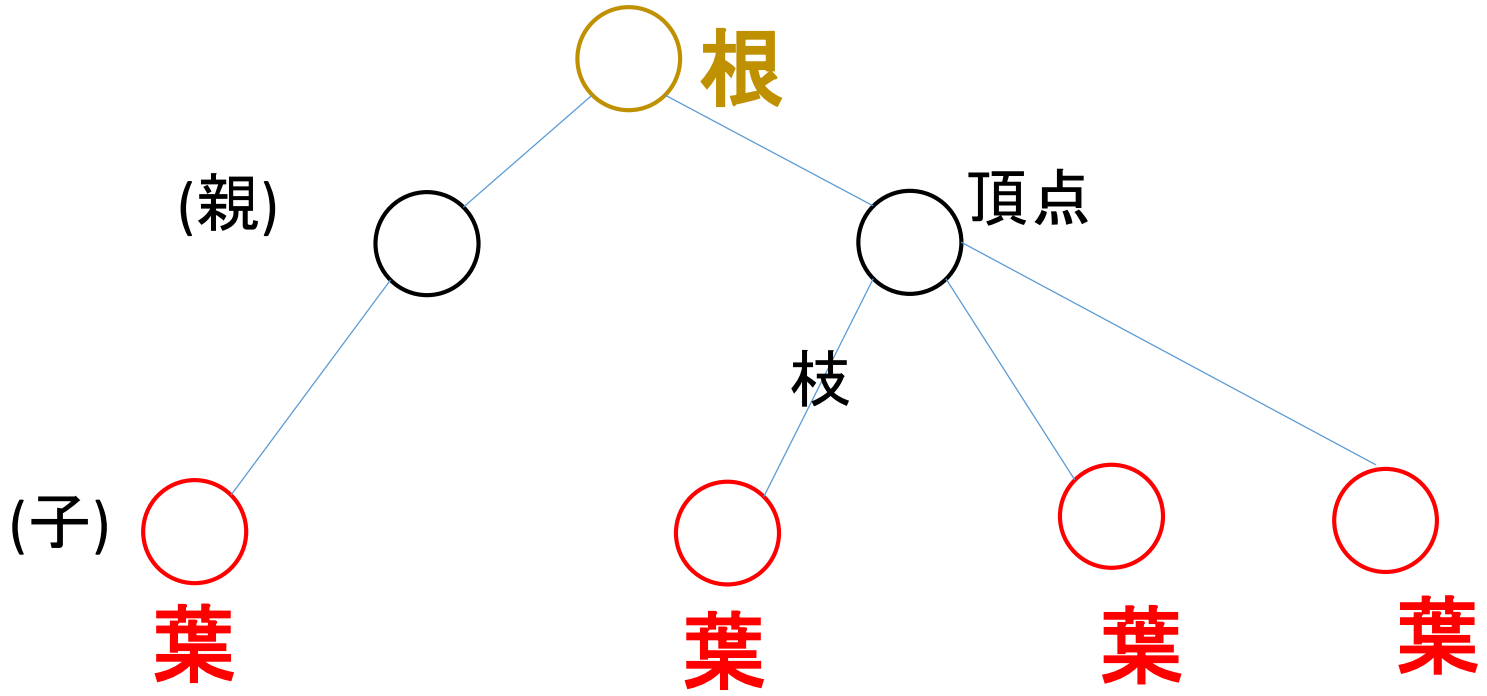
$$23 + \left(\frac{15 + 30}{7 + 2 \cdot 4}\right) \times (8 + 13)$$

二つの数の加減乗除しか知らない小学生にこの計算の仕方を教えよう！

1.文章の意味を理解する(2)

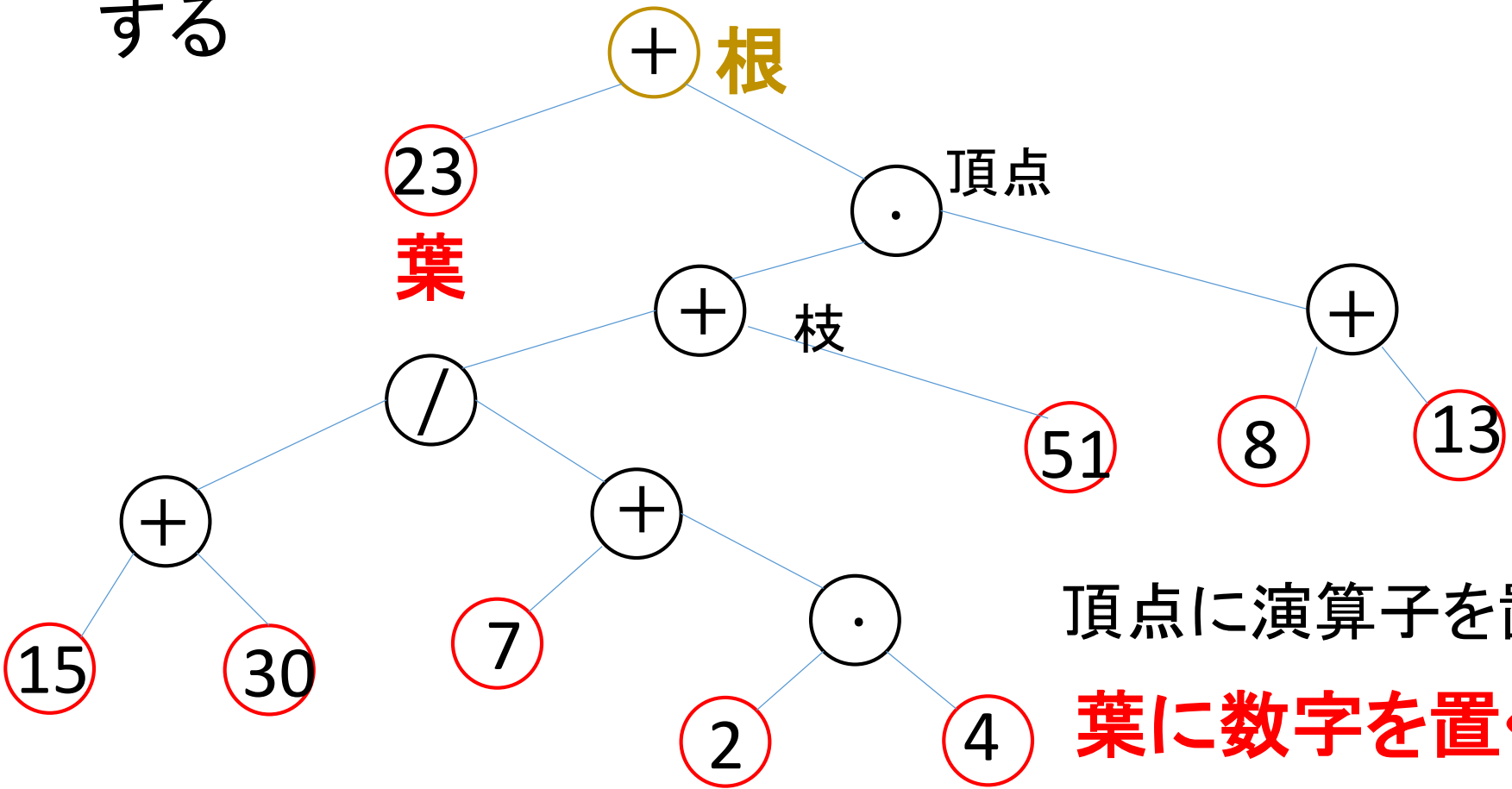
木をつかって数式(文章)を記述する

木＝連結で、閉路のない、無向グラフ



1.文章の意味を理解する(3)

木をつかって$23 + \left(\frac{15+30}{7+2\cdot4} + 51\right) \times (8 + 13)$を記述する

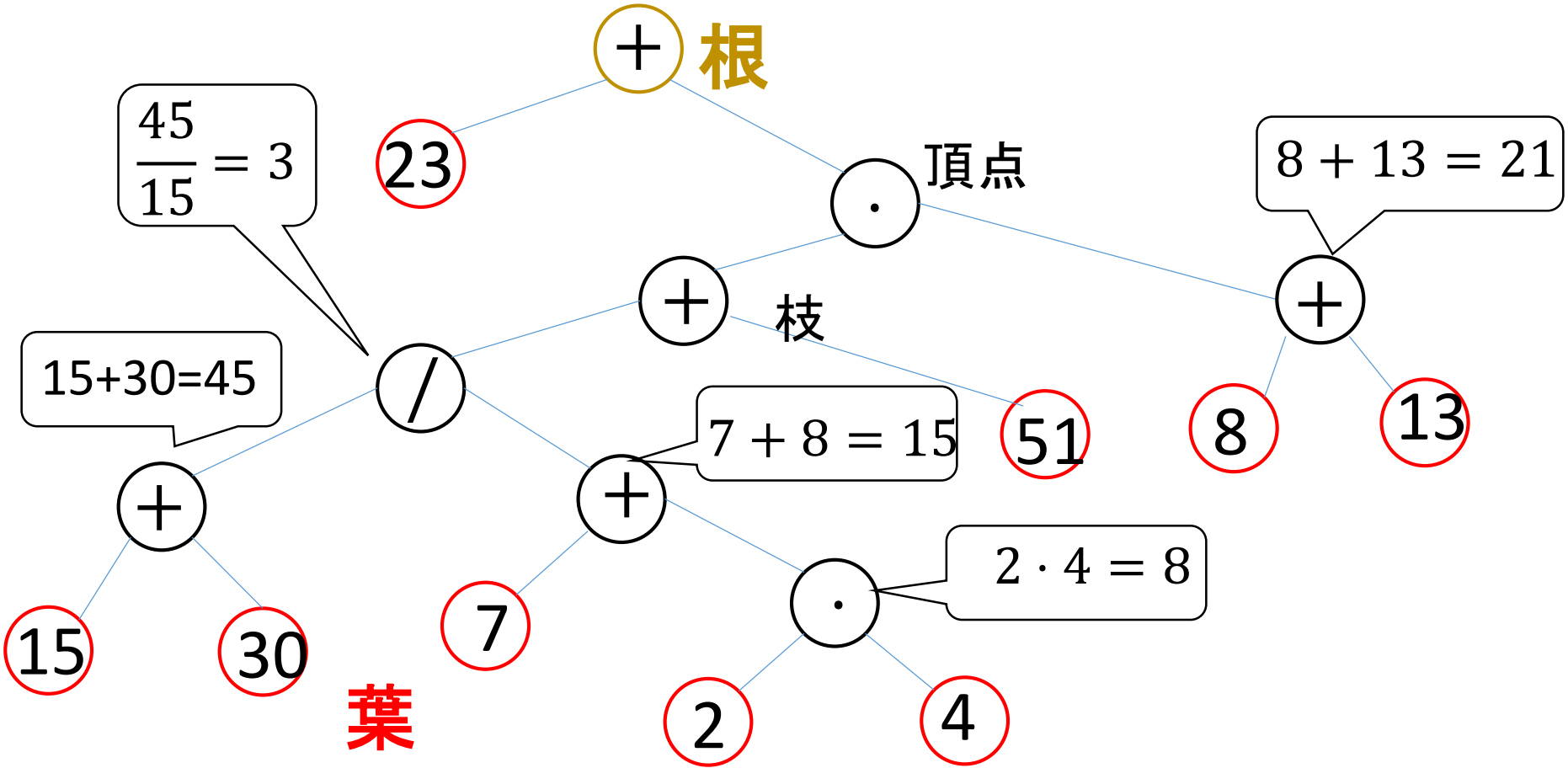


頂点に演算子を置く

葉に数字を置く

1.文章の意味を理解する(3)

葉から初めて、子から親へ計算をしていく



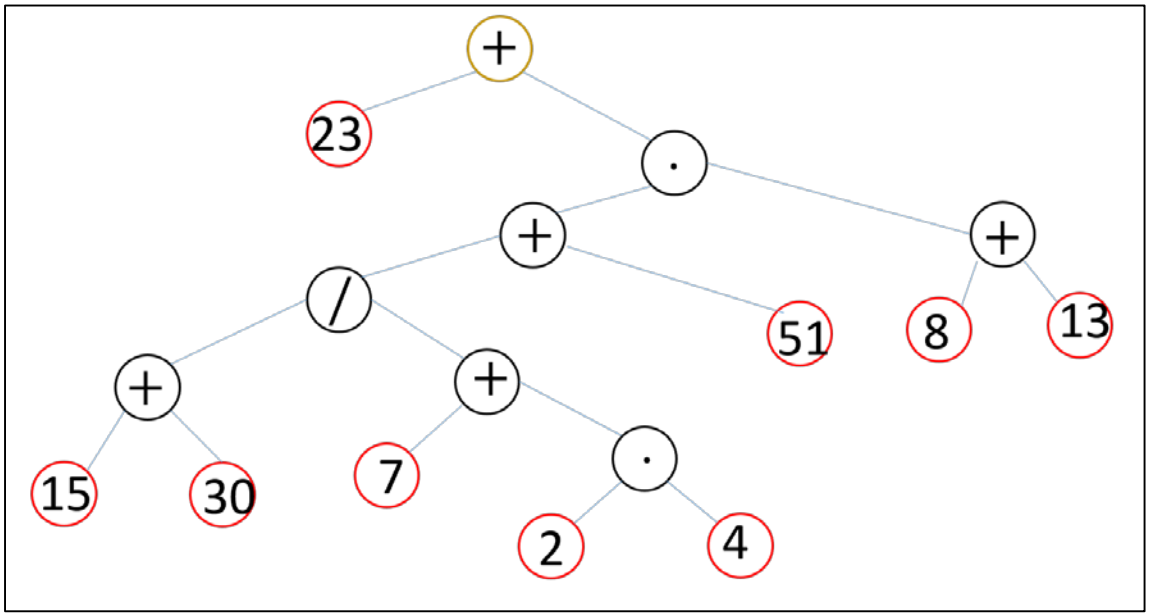
1.文章の意味を理解する(4)

文章: $23 + \left(\frac{15+30}{7+2\cdot 4} + 51\right) \times (8 + 13)$

対応する意味:木

木があれば、式の値を簡単に計算できる

⇔導出木と呼ぶ



文章から木を導くルールを文法と呼ぶ

2.文章の正しさを解析する(1)

文章の意味が定まるためには、文章が『正しい』必要がある。

⇒正しい文章ってなに？

例：カッコ文

数式から数値や演算子を消しカッコだけ残した。

(( ) (( )) (( ))

この文章は正しいだろうか？

2.文章の正しさを解析する(2)

例：カッコ文
数式から数値や演算子を消しカッコだけ残した。
(( ) (( )) (( ))
この文章は正しいだろうか？

答え：正しくない
理由：左カッコ ”(“ が 右カッコ ”)” よりも多いから。
(6個) (5個)
左カッコの数＝右カッコの数 が必要

2.文章の正しさを解析する(3)

カッコ文の二つ目の例:

(( ) ((( )))) (( ) (( ))

この文章は正しいだろうか？

2.文章の正しさを解析する(3’)

カッコ文の二つ目の例:

(( ) ((( )))) (( ) (( ))

この文章は正しいだろうか？

答え:正しくない

(( ) ((( )))) (( ) (( ))

赤いところだけをみると、右カッコ”)”が左カッコ”(”よりも多い。

既出のカッコが全て閉じられているのに、次に右カッコ”)”が来ている

2.文章の正しさを解析する(4)

結局：カッコ文は次の条件を満たすときに「正しい」

・文全体に対して：

左カッコ ”(” の数　＝　右カッコ ”)” の数

・任意のnに対して、最初からn文字目までで、

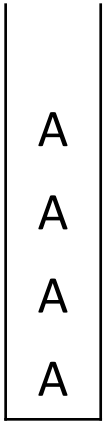
左カッコ ”(” の数　≥　右カッコ ”)” の数

2.文章の正しさを解析する(4')

練習：カッコ文の正しさを検証するにはどうしたら良いか考えよ。

(どのようなプログラムを書けばよいか？)

ヒント：スタックを使うと簡単



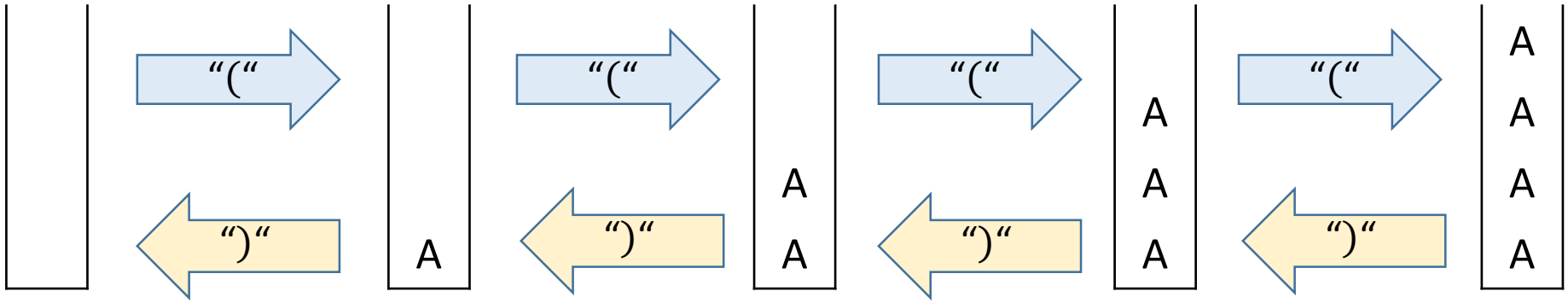
2.文章の正しさを解析する(5)

解答例：カッコ文を左から読んでいく。

左カッコを読む　⇒　スタックを一つ積む

右カッコを読む　⇒　スタックを一つ取り出す

- スタックが空のとき右カッコを読むと、『正しくない』と判断
- 最後まで文を読んだとき、スタックが空なら正しいと判断



プッシュダウンオートマトンの簡単な例になっている

3.形式言語の導入(1)

形式言語:　(プログラミング or 人工)言語の(数学的)モデル

文字 (もしくは記号)

日本語:ひらがな、カタカタ、漢字

形式言語:0,1,a,b,c,... などの数字やローマ字

アルファベット

⇔記号の有限集合

例:$\{0,1\}, \{a, b, c\}$ など

この授業では、アルファベットをΣ (シグマ)で通常表す。

3.形式言語の導入(2)

アルファベット Σ 上の列(もしくは文章もしくは記号列)

$\Leftrightarrow$ Σ上の記号を重複を許して一列に並べたもの

$\Leftrightarrow$ ある$n \in \mathbb{N}$に対して、$\Sigma^n := \Sigma \times \cdots \times \Sigma$の元

例:

$\Sigma = \{0,1\}$のとき、0, 1, 01, 10, 1011, 10010011, …などが列

$\Sigma = \{a, b, c\}$のとき、

$a,\ b,\ c,\ ab, ba, bc, \cdots, abcba, \cdots$ などが列

3.形式言語の導入(3)

列xの長さ $|x|$:= 列に含まれる記号の個数

例:

$\Sigma = \{0,1\}$ のとき、　$x = 01101$の長さは、$|x| = 5$

$\Sigma = \{a, e, s\}$ のとき、　$x = sea$の長さは、$|x| = 3$

空列ϵ := 長さが0の列

よって$|\epsilon| = 0$

3.形式言語の導入(4)

列xの逆x^R ≔ xを後ろから順に読んだ列

- $x = x_1 x_2 \cdots x_{n-1} x_n$ $(x_i \in \Sigma)$のとき
  $x^R = x_n x_{n-1} \cdots x_2 x_1$
- $x = a$ $(a \in \Sigma)$のとき、$x = x^R = a$
- $\epsilon^R \coloneqq \epsilon$

例:

$x = 01011$, $y = 00$, $z = 1101$なら、

$x^R = 11010, y^R = 00, z^R = 1011$

3.形式言語の導入(5)

列xと列yの連接xy := 列 xの後に列yをつけた列

例:$x = 1101, y = 000$なら　$xy = 1101000$

同じ列の連接:　$x^2 := xx,$

例:$x = 10$のとき、$x^2 = 1010,$

空列ϵと任意の列xに対して、$\epsilon x = x\epsilon = x$

3.形式言語の導入(6)

3つ以上の列の連接

列x, y, zに対して、$xyz \coloneqq (xy)z = x(yz)$

$x^3 \coloneqq (x^2)x = xxx, \cdots,$　　$x^n = (x^{n-1})x = \overbrace{x \cdots x}^{n}$

$x^0 \coloneqq \epsilon$と約束しておく。

例：$x = 10$のとき、
$x^2 = 1010, x^3 = 101010, x^4 = 10101010,$

☆連接は結合則を満たす。

3.形式言語の導入(7)

列xの部分列：列xの一部分の列のこと

列wは列xの部分列　⇔　ある列y, zが存在して$x = ywz$

例、

$x = 10011$のとき、

$y = 0, z = 01, w = 1001$とすると、

y、z、wはすべてxの部分列である。

3.形式言語の導入(8)

アルファベットΣ上の長さnの列の集合：Σ^n

$$\Sigma^n = \overbrace{\Sigma \times \cdots \times \Sigma}^{n}$$

$\Sigma^0 := \{\epsilon\}$ と定義する。

アルファベットΣ上の全ての列の集合：Σ^*、

$$\Sigma^* := \bigcup_{n \geq 0} \Sigma^n = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \cdots$$

3.形式言語の導入(9)

アルファベットΣ上の言語L : ＝ Σ上の列の集合

つまり、Σ^*の部分集合　($L \subset \Sigma^*$)

$\{0,1\}$上の言語の例:

有限集合:$\{0,01,000,10101\}$

無限集合:$\{1^n 0^n \mid n \geq 0\}$　　10, 1100, 111000,....という集合

『無限集合をいかにして書き表すか』が前半の主題

言語の特別な例:

全ての列の集合:Σ^*、例えば$\{0,1\}^* := \{\epsilon, 0,1,00,01,10, \ldots,\}$

空集合$\emptyset$: どのような列も含まない言語

空列集合$\{\epsilon\}$: 空列ϵのみからなる言語

3.形式言語の導入(10)

例題

以下はどのような言語であるか推測しなさい。

$$L_1 = \{1, 010, 00100, 0001000, 000010000, \cdots\}$$

$$L_2 = \{\epsilon, 00, 11, 0000, 0101, 1010, 1111, 000000, 001001, 010010, 011011, \cdots\}$$

$$L_3 = \{1, 010, 011, 110, 111, 00100, 00101, 00110, 00111, 01100, 01101, 01110, 01111, 10100, 10101, 10110, 10111, 11100, 11101, 11110, 11111, 0001000, 0001001, \cdots\}$$

3.形式言語の導入(11)

解答例

$$L_1 = \{1, 010, 00100, 0001000, 000010000, \cdots\}$$
$$= \{0^n 10^n \mid n \geq 0\}$$

☆この授業ではnは整数にしか用いないとする。

$$L_2 = \{\epsilon, 00, 11, 0000, 0101, 1010, 1111, 000000, 001001, 010010, 011011, \cdots\}$$
$$= \{xx \mid x \in \Sigma^*\}$$

$$L_3 = \{1, 010, 011, 110, 111, 00100, 00101, 00110, 00111, 01100, 01101, 01110, 01111, 10100, 10101, 10110, 10111, 11100, 11101, 11110, 11111, 0001000, 0001001, \cdots\}$$
$$= \{x1y \mid x, y \in \Sigma^*, |x| = |y|\}$$

3.形式言語の導入(12)

例題1.2　列$x \in \Sigma^*$の逆をより形式的に定義せよ。

解答

再帰的定義を用いる。

逆x^Rは、以下の(i)と(ii)により再帰的に定義される:

(i)　$\epsilon^R = \epsilon$

(ii)　列$x \in \Sigma^*$と記号(長さ1の列)$a \in \Sigma$に対して、$(ax)^R = x^R a$

例:$(abc)^R$に上記の定義を適応する

$$(abc)^R = \big(a(bc)\big)^R = (bc)^R a = \big(b(c)\big)^R a = (c)^R ba$$
$$= \big(c(\epsilon)\big)^R ba = \epsilon^R cba = \epsilon cba = cba$$

3.形式言語の導入(13)

例題

列xで$x = x^R$を満たすものを回文と呼ぶ。
回文の再帰的定義を与えよ。

ヒント：回文を再帰的に作成するルールを考える

3.形式言語の導入(14)

例題

列xで$x = x^R$を満たすものを回文と呼ぶ。
回文の再帰的定義を与えよ。

（回文の例：ϵ, 0,11,101, 000,0110, 01010 など）

解答

(i)　空列ϵと長さ1の列（記号）は回文である。

(ii)　回文wと記号$x \in \Sigma$に対して、列xwxは回文である。

(i)と(ii)を用いて生成できるものだけが回文である。

3.形式言語の導入(15)

例題1.3 xとyを列としたとき、$(xy)^R = y^R x^R$を証明せよ。

解答

列xyの長さに関する数学的帰納法で証明する。

(i) $|xy| = 0$のとき、$x = y = \epsilon$なので、
$(\epsilon\epsilon)^R = (\epsilon)^R = \epsilon = \epsilon\epsilon = \epsilon^R\epsilon^R$で正しい。

3.形式言語の導入(16)

(ii) $|xy| = n$のとき命題が正しいと仮定する。

$|xy| = n + 1$のときを考える:

$x = \epsilon$のとき、$(\epsilon y)^R = y^R = y^R\epsilon = y^R\epsilon^R$でOK

$x \neq \epsilon$のとき、ある$a \in \Sigma$が存在して、$x = ax'$と書ける。
$x'y$は長さnなので、帰納法の仮定より$(x'y)^R = y^R x'^R$
よって$(xy)^R = (ax'y)^R = \big(a(x'y)\big)^R = (x'y)^R a = y^R x'^R a$

一方、$y^R x^R = y^R (ax')^R = y^R x'^R a$ なのでOK

(i) (ii)よりすべてのnに対して命題は証明された。

3.形式言語の導入(17)

言語L_1とL_2の連接L_1L_2

$$L_1L_2 := \{x_1x_2 \mid x_1 \in L_1, x_2 \in L_2\}$$

例:

$L_1 = \{00,100\}, L_2 = \{1111,001\}$のとき、

$$L_1L_2 = \{001111, 00001, 1001111, 100001\}$$

空列のみの言語$\{\epsilon\}$:

任意の言語Lに対して、$\{\epsilon\}L = L\{\epsilon\} = L$

3.形式言語の導入(18)

例題1.4 任意の言語Lに対して、$L\emptyset = \emptyset L = \emptyset$を示せ。

解答: $L\emptyset = \emptyset$を証明する。$\emptyset L = \emptyset$は同様に証明できる。

任意の列xに対して、

$x \in L_1 L_2 \Leftrightarrow \exists x_1 \exists x_2 (x = x_1 x_2 \text{ and } x_1 \in L_1 \text{ and } x_2 \in L_2)$

である。

今、$L_2 = \emptyset$なので$x_2 \in L_2$となるx_2は存在しない。
したがって、"$x = x_1 x_2 \text{ and } x_1 \in L_1 \text{ and } x_2 \in L_2$"を満たす$x_1$と$x_2$の組は存在しない。$L_1 L_2 = L\emptyset$に入る$x$は存在しない。

よって、$L\emptyset$は空集合である。

3.形式言語の導入(19)

$L^2 = LL, L^3 = LLL,$ と定義する。一般に$L^n = LL^{n-1}$

特別な場合として、$L^1 = L$, $L^0 = \{\epsilon\}$と定義する。

言語Lのクリーネ閉包L^*:

$$L^* = \bigcup_{n \geq 0} L^n = L^0 \cup L^1 \cup L^2 \cdots$$

“$*$”のことをスター演算と呼ぶ。

Σ上の全ての列の集合Σ^*は、Σを長さ1の列すべてからなる言語と考えると、言語Σのクリーネ閉包になっている。

3.形式言語の導入(20)

例題 1.6

長さが奇数の$\{0,1\}$上の列全体からなる言語を表現せよ

解答例1:

長さが偶数の列全体は$\{00,01,10,11\}^*$である。これに0か1を一つ加えて:

$$\{00,01,10,11\}^*\{0,1\}$$

解答例2:全ての列の集合から長さが偶数の列の集合を引いて:

$$\{0,1\}^* - \{00,01,10,11\}^*$$

3.形式言語の導入(21)

鳩ノ巣原理：鳩が$n+1$羽いて巣箱がn個しかないなら、必ず一つの巣箱には2羽以上入らなければならない。

あたりまえのことを言っているように思うが、

今後、様々な定理の証明で重要な役割を果たす。

3.形式言語の導入(22)

例題 1.7 nを1以上の整数とし、集合Aはその大きさが$n+1$で、$A \subseteq \{1,2,\cdots,2n\}$を 満たすとする。このとき、 A の元x_1, x_2 で、x_1がx_2を割り切るものが存在することを示せ。

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、$a_i = 2^{b_i} \cdot p_i$と書く。

ここでp_iは奇数。よって、$p_i \in \{1,3,\cdots,2n-1\}$

集合$\{1,3,\cdots,2n-1\}$の大きさはnなのに、Aの大きさは$n+1$なので、あるp_iとp_j $(i \neq j)$ は、$p_i = p_j$を満たす。

よって、$a_i = 2^{b_i} \cdot p_i$と$a_j = 2^{b_j} \cdot p_i$と書ける。

よって、$b_i \geq b_j$ならa_iがa_jで、$b_j > b_i$ならa_jがa_iで割り切れる。

第２章：正規表現と有限オートマトン

1. 正規表現
2. 有限オートマトン
3. 非決定性有限オートマトン
4. 有限オートマトンと正規表現の等価性
5. 有限オートマトンの能力の限界

1.正規表現(1)

言語（列の集合）を表現する方法の一つ (こっちがオリジナル)

アルファベットΣ上の正規表現S、およびSが表現する言語$L(S)$は以下のように再帰的に定義される：

① $\emptyset$は正規表現であり、$L(\emptyset) = \emptyset$.

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

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

④ R_1とR_2が正規表現なら、$(R_1 + R_2)$, (R_1R_2), (R_1^*)も正規表現であり、$L\big((R_1 + R_2)\big) = L(R_1) \cup L(R_2)$,
$L\big((R_1R_2)\big) = L(R_1)L(R_2)$, $L(R_1^*) = \big(L(R_1)\big)^*$である。

1.正規表現(2)

Σ上の正規表現S、およびSが表現する言語$L(S)$の定義:

① $\emptyset$は正規表現であり、$L(\emptyset) = \emptyset$.

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

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

④ R_1とR_2が正規表現なら、$(R_1 + R_2)$, (R_1R_2), (R_1^*)も正規表現であり、$L\big((R_1 + R_2)\big) = L(R_1) \cup L(R_2)$,
$L\big((R_1R_2)\big) = L(R_1)L(R_2)$, $L(R_1^*) = \big(L(R_1)\big)^*$である。

(例)$\Big(\big(0(1^*)\big) + 0\Big)$: まず、0は③より正規表現. 次に、$1^*$は④より正規表現. よって、$0(1^*)$も④より正規表現. 最後に、全体が④より正規表現になる。

1.正規表現(3)

① $L(\emptyset) = \emptyset$. ② $L(\epsilon) = \{\epsilon\}$. ③ $a \in \Sigma$なら、$L(a) = \{a\}$.

④$L\big((R_1 + R_2)\big) = L(R_1) \cup L(R_2)$,

$L\big((R_1 R_2)\big) = L(R_1)L(R_2)$, $L(R_1^*) = \big(L(R_1)\big)^*$.

記法の省略:

$\big((0(1^*)) + 0\big)$を$01^* + 0$と書く。

連接と+は結合則が成り立つので、

$\big((0 + 1) + 2\big)^*$を$(0 + 1 + 2)^*$と書く。

$R^+ := RR^*$, $R^2 = RR, \cdots, R^5 = RRRRR, \cdots$なども用いる。

1.正規表現(4)

① $L(\emptyset)=\emptyset$. ② $L(\epsilon)=\{\epsilon\}$. ③ $a\in\Sigma$なら、$L(a)=\{a\}$.
④$L\big((R_1+R_2)\big)=L(R_1)\cup L(R_2)$,
$L\big((R_1R_2)\big)=L(R_1)L(R_2)$, $L(R_1^*)=\big(L(R_1)\big)^*$.

例題2.1 次の正規表現はどのような言語を表現しているか:

(i) $R_1=\emptyset^*$ (ii) $R_2=\big((0+\epsilon)^*+(001+11)^*\big)\emptyset$
(iii) $R_3=(0+1)^*000(0+1)^*$

(解答) (i) 正規表現のルール④より、$L(R_1)=L(\emptyset)^*=\emptyset^*$.
$\emptyset^*=\emptyset^0+\emptyset^1+\emptyset^2+\cdots$ だが、$\emptyset^0=\{\epsilon\}$, $\emptyset^i=\emptyset\ (i\geq 1)$より、

$$L(R_1)=\emptyset^*=\{\epsilon\}$$

(ii) $R=\big((0+\epsilon)^*+(001+11)^*\big)$とすると、$R_2=R\emptyset$と書ける。
ルール④と①より、$L(R_2)=L(R\emptyset)=L(R)L(\emptyset)=L(R)\emptyset=\emptyset$.

(iii) $L\big((0+1)^*\big)=\{0,1\}^*$となり、すべての列の集合になる。よって、$L(R_3)=\{0,1\}^*\{000\}\{0,1\}^*$となり、どこかに3個以上の0が連続して現れる$\{0,1\}$上の列の全体となる。

1.正規表現(5)

例題2.2 $R_4 = c^*(a + bc^*)^*$はどのような言語を表現しているか。

(解答) $L(R_4)$は部分列acを含まない$\{a, b, c\}$上の列全体になる。

証明の概略:

$$L(R_4) = L(c^*(a + bc^*)^*) = L(c^*)\left(L(a + bc^*)\right)^*$$
$$= \{c\}^*\{a, b, bc, bcc, bccc, \cdots\}^*$$

よって、$A = \{a, b, bc, bcc, bccc, \cdots\}$とすると、$x \in L(R_4)$ならば、

$$x = c^j y_1 y_2 \cdots y_i \cdots y_n, \qquad y_i \in A$$

- c^jにはacは現れない。
- c^jとy_1の境界でもacは現れない。
- y_iの中では、acは現れない。
- 最後にy_iとy_{i+1}の境界でも現れない。

よって、$x \in L(R_4)$ならばxはacを部分列に含まない。

1.正規表現(6)

例題2.2 $R_4 = c^*(a + bc^*)^*$はどのような言語を表現しているか。

(解答) $L(R_4)$は部分列acを含まない$\{a, b, c\}$上の列全体になる。

証明の概略:

次にx がacを部分列に含まないならば、$x \in L(R_4)$を証明する。

x がacを部分列に含まないと仮定する。

列xを左から見て行って、記号cの前以外(つまりaの前とbの前)と列全体の前と後ろに『切目』を入れる:

例えば、$x = |ccc|a|a|bcc|b|bc|a|bccc|b|b|$

1.正規表現(7)

x がacを部分列に含まないと仮定する。

列xを左から見て行って、記号cの前以外(つまりaの前とbの前)と列全体の前と後ろに『切目』を入れる:

今、$x = \cdots |y| \cdots$, と書けたとすると、以下の場合が考えられる:

(yの右端がaの場合) aの左にも切れ目が入り$y = a$

(yの右端がbの場合) bの左にも切れ目が入り$y = b$

(yの右端がcの場合) このcから左に見ていくと、cの左には切れ目が入らないし、次がまたcでも同様。もしcの左にbが来るとその左に切目が入る。仮定よりcの左にはaは来ない。
よって、$y = bc^i \ (i \geq 1)$と書ける。

１.正規表現(８)

x がacを部分列に含まないと仮定する。

列xを左から見て行って、記号cの前以外(つまりaの前とbの前)と列全体の前と後ろに『切目』を入れる:

$x = \cdots |y| \cdots$, と書けたとする。

・・・・

・・・・

いずれの場合も$y \in A = \{a, b, bc, bcc, bccc, \cdots\}$なので、結局、$x \in \{c\}^* A^* = L(c^*(a + bc^*)^*)$となる。

■

1.正規表現(9)

$x \in \{0,1\}^*$にたいして、$\#_1(x)$を列xに含まれる1の個数と定義。
$\#_0(x)$も同様。

例題2.3　$L(R) = \{x \mid x \in \{0,1\}^*$かつ$\#_1(x)$が偶数$\}$の正規表現を求めよ。

(解答) 例えば、$x = 0010010111101000 \in L(R)$は、
$x = 0^2(10^210^1)(10^010^0)(10^110^3)$と書ける。

よって、$R = 0^*(10^*10^*)^*$が$L(R)$に対応する正規表現。

1.正規表現(10)

例題:
$L_1 := \{x \in \{0,1\}^* |$かつ$\#_1(x) = \#_0(x)$かつ
xの任意のプレフィックスyに対して、$0 \leq |\#_1(y) - \#_0(y)| \leq 1\}$
の正規表現を求めよ。

ただし、プレフィックスとは列の先頭から始まる部分列のこと:
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。

(解答例)

xが0から始まるとする。$|\#_1(00) - \#_0(00)| = 2$なので、

0の次は1になる。

$|\#_1(01) - \#_0(01)| = 0$なので、その次は0でも1でもよい。

もし$x = 010\cdots$なら、最後の0の次は、1でなければならない。

同様に、$x = 011\cdots$なら、最後の1の次は、0でなければならない。

結局、正規表現は、$(01 + 10)^*$と書ける。

1.正規表現(11)

練習問題:

言語$L_2 := \{x \in \{0,1\}^* |\ \#_1(x) = \#_0(x)$かつ
xの任意のプレフィックスyに対して、$0 \leq |\#_1(y) - \#_0(y)| \leq 2\}$

の正規表現を求めよ。

ただし、プレフィックスとは列の先頭から始まる部分列のこと: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。

ヒント

$L_1 := \{x \in \{0,1\}^* |$かつ$\#_1(x) = \#_0(x)$かつ
xの任意のプレフィックスyに対して、$0 \leq |\#_1(y) - \#_0(y)| \leq 1\}$
の正規表現は、$(01 + 10)^*$と書けることを使う。

L_1を使って、L_2がどのように書けるかを考えるとよい。

例えば$x = 1\,1\,0\,1\,0\,0\,1\,1\,0\,0$ は$1L_10$と書ける。

1.正規表現(12)

$L_2 := \{x \mid x \in \{0,1\}^*$かつ$x$の任意のプレフィックス$y$
に対して、$0 \leq |\#_1(y) - \#_0(y)| \leq 2\}$
$L_1 := \{x \mid x \in \{0,1\}^*$かつ$x$の任意のプレフィックス$y$
に対して、$0 \leq |\#_1(y) - \#_0(y)| \leq 1\}$
の正規表現は、$(01 + 10)^*$と書ける。
（解答例）まず、具体的な 例で考える。

x: $1\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 0\ 0 \quad \in 1L_1 0$
$\#_1(y) - \#_0(y)$: $1\ \underbrace{2\ 1\ 2\ 1\ 0\ 1\ 2\ 1}_{L_1}\ 0$

x: $1\ 1\ 0\ 0 \quad 0 \quad 0 \quad 1\ 1\ 1\ 0 \quad \in 1L_1 00L_1 1L_1$
$\#_1(y) - \#_0(y)$: $1\ \underbrace{2\ 1}\ 0\ -1\ \underbrace{-2\ -1}\ \underbrace{0\ 1}\ 0$

x: $1\ 1\ 0\ 1\ 0\ 1\ 0\ 0 \quad 0 \quad 0 \quad 1\ 1 \quad \in 1L_1 00L_1 1$
$\#_1(y) - \#_0(y)$: $1\ \underbrace{2\ 1\ 2\ 1\ 2\ 1}\ 0\ -1\ \underbrace{-2\ -1}\ 0$

1.正規表現(13)

$L_2 := \{x \mid x \in \{0,1\}^*$かつ$x$の任意のプレフィックス$y$
に対して、$0 \leq |\#_1(y) - \#_0(y)| \leq 2\}$
$L_1 := \{x \mid x \in \{0,1\}^*$かつ$x$の任意のプレフィックス$y$
に対して、$0 \leq |\#_1(y) - \#_0(y)| \leq 1\}$
の正規表現は、$(01 + 10)^*$と書ける。
（解答例）

x:	1 1 0 0 0 0 1 1 1 0	$\in 1L_1 00 L_1 1 L_1$
$\#_1(y) - \#_0(y)$:	1 2 1 0 −1 −2 −1 0 1 0	

x:	1 1 0 1 0 1 0 0 0 0 1 1	$\in 1L_1 00 L_1 1$
$\#_1(y) - \#_0(y)$:	1 2 1 2 1 2 1 0 −1 −2 −1 0	

結局、$L_2 = (1L_1 0 + 0L_1 1)^* L_1$
$= (1(10 + 01)^* 0 + 0(10 + 01)^* 1)^* (10 + 01)^*$

1.正規表現(14)

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^*もまた正規表現で表現できる。

（証明）正規表現の定義：

④　R_1とR_2が正規表現なら、$(R_1 + R_2)$, (R_1R_2), (R_1^*)も正規表現であり、$L\big((R_1 + R_2)\big) = L(R_1) \cup L(R_2)$,
$L\big((R_1R_2)\big) = L(R_1)L(R_2)$,　$L(R_1^*) = \big(L(R_1)\big)^*$である。

より定理は自明である。 ■

問題

以下の正規表現Rに対して、Rが表現する言語$L(R)$に含まれる文字列を2つ、$L(R)$に含まれない文字列を2つそれぞれ挙げよ。

① a^*b^*

② $a(ba)^*b$

③ $a^* + b^*$

④ $(aaa)^*$

⑤ $(a+b)^*a(a+b)^*b(a+b)^*a(a+b)^*$

⑥ $aba + bab$

⑦ $(\epsilon + a)b$

⑧ $(a + ba + bb)(a+b)^*$

問題

以下の言語に対する正規表現を与えよ。ただし、アルファベットは$\{0,1\}$とする。(つまり$w \in \{0,1\}^*$):

① $\{w \mid w$は1で始まり0で終わる$\}$

② $\{w \mid w$は少なくとも三つの1を含む$\}$

③ $\{w \mid w$は部分列0101含む$\}$

④ $\{w \mid 3 \leq |w|$ かつ wの3番目の文字が0 $\}$

⑤ $\{w \mid w$は 0で始まり長さが奇数、
もしくは、1で始まり長さが偶数$\}$

⑥ $\{w \mid w$は部分列110を含まない$\}$

⑦ $\{w \mid |w| \leq 5\}$

問題(前ページの続き)

⑧ $\{w \mid w \neq 11$ かつ $w \neq 111\}$

⑨ $\{w \mid w$の全ての奇数番目は$1\}$

⑩ $\{w \mid \#_1(w) \geq 2$ かつ$\#_0(w) \geq 1\}$

⑪ $\{0, \epsilon\}$

⑫ $\{w \mid \#_0(w)$が偶数、もしくは$\#_1(w) = 2\}$

⑬ $\emptyset$

⑭ $\{w \mid w \neq \epsilon\}$

⑮ $\{w \mid |w|$が偶数、もしくは$|w|$が3の倍数$\}$

2.有限オートマトン (1)

正規表現 ⇔ 言語に属する列を「生成」する

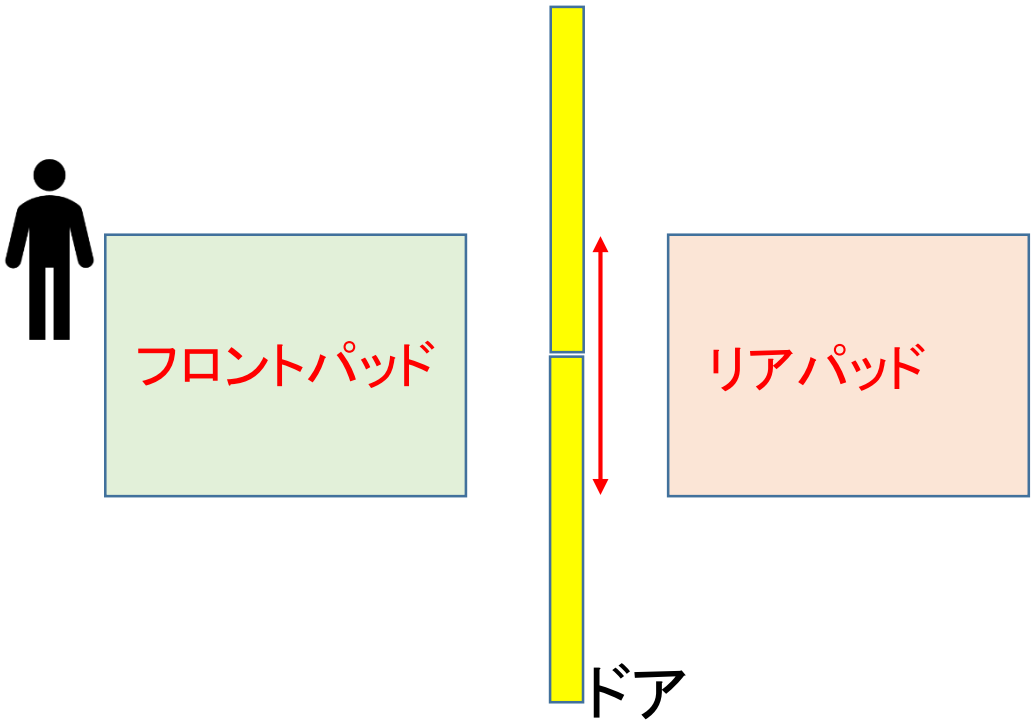
有限オートマトン ⇔ 与えられた列が言語に属するかどうかを
「判定する機械」

ただし、あくまでも「イメージ」で、
オートマトンは言語を表現するための「数学的手段」である。

2.有限オートマトン(1－1)

有限オートマトン＝メモリ容量が著しく制限された
計算機のモデル

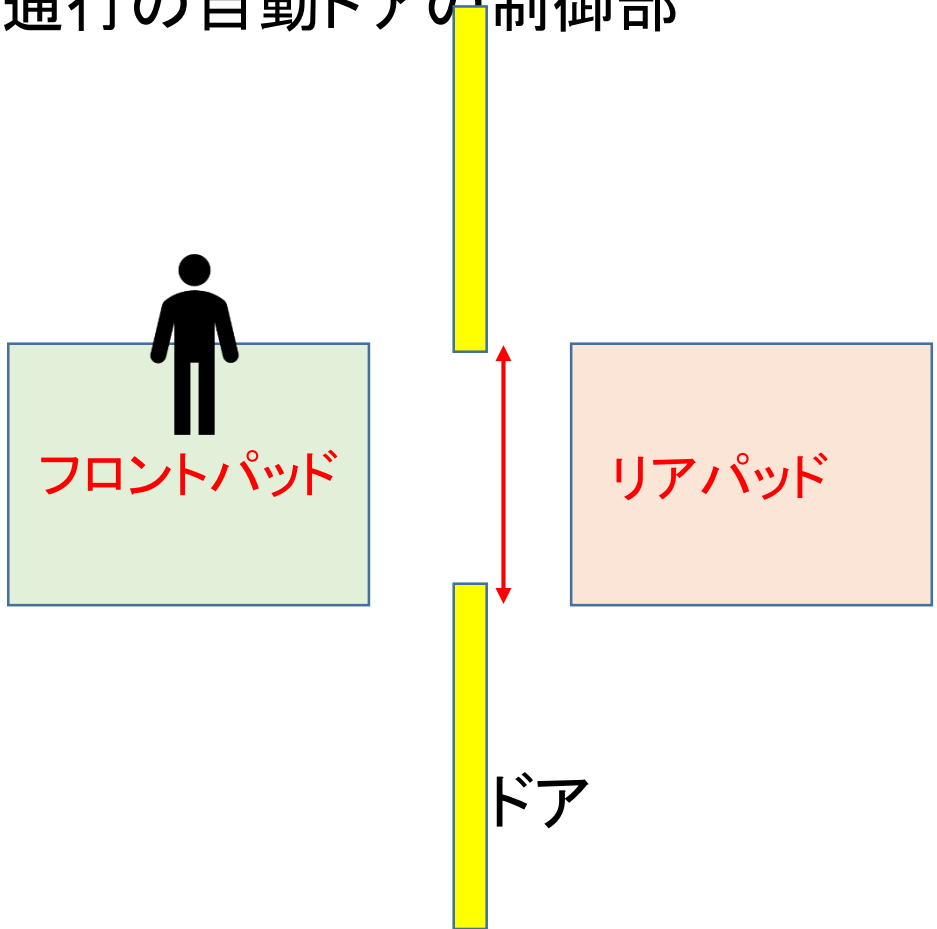
例） 一方通行の自動ドアの制御部



2.有限オートマトン(1－1)

有限オートマトン＝メモリ容量が著しく制限された
計算機のモデル

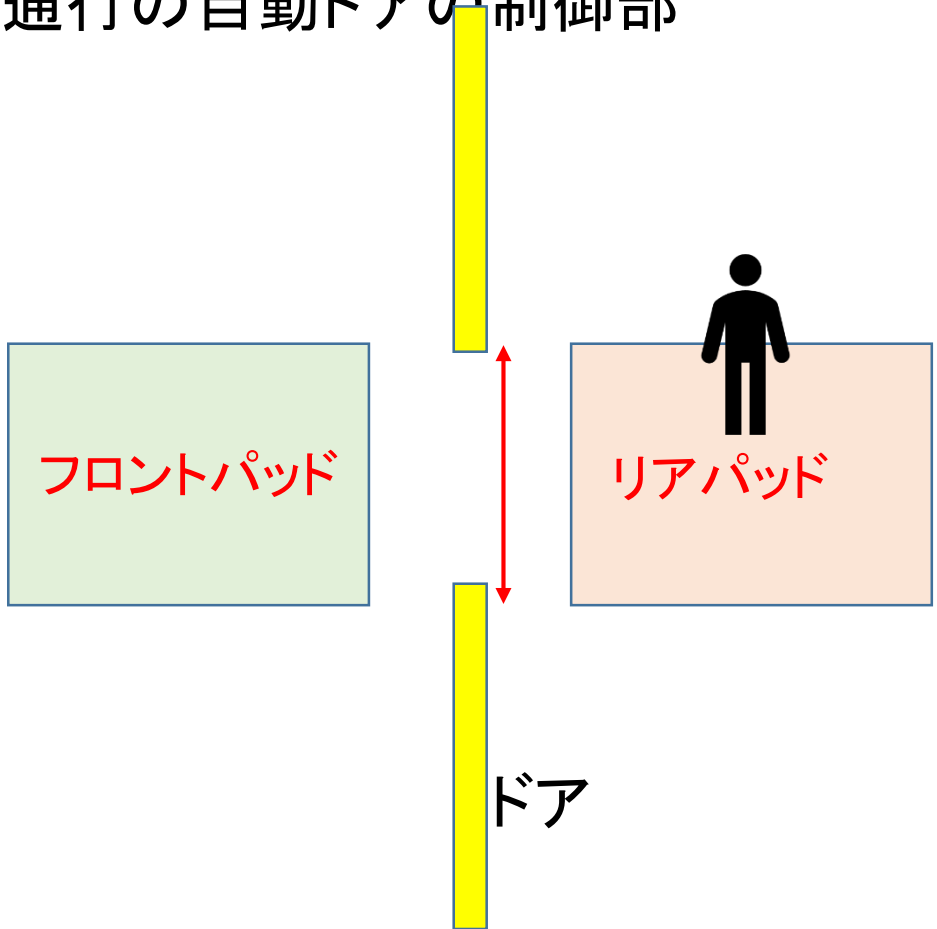
例） 一方通行の自動ドアの制御部



2.有限オートマトン(1－1)

有限オートマトン＝メモリ容量が著しく制限された
計算機のモデル

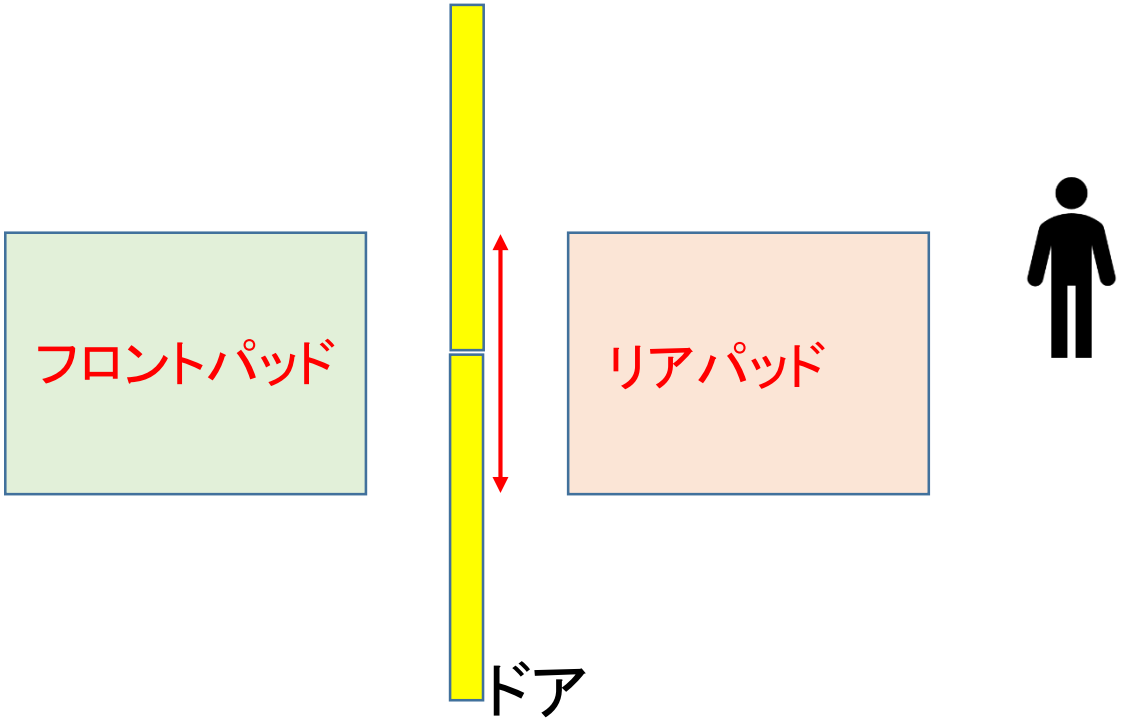
例） 一方通行の自動ドアの制御部



2.有限オートマトン (1－1)

有限オートマトン＝メモリ容量が著しく制限された
計算機のモデル

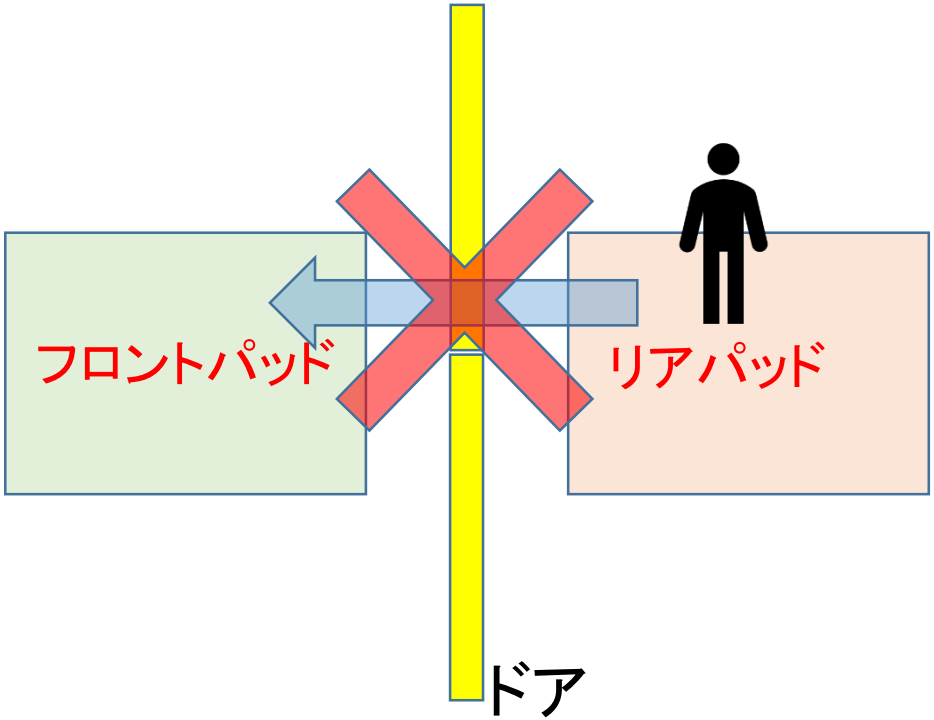
例） 一方通行の自動ドアの制御部



2.有限オートマトン (1－1)

有限オートマトン＝メモリ容量が著しく制限された
計算機のモデル

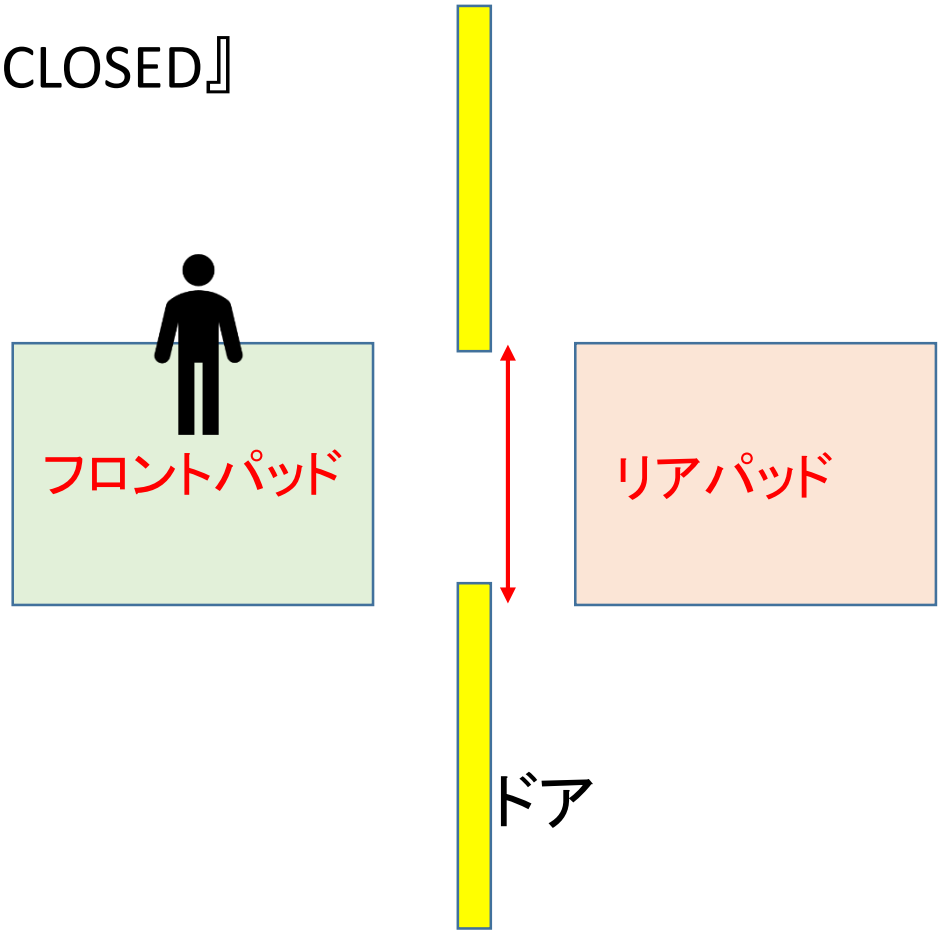
例） 一方通行の自動ドアの制御部



2.有限オートマトン (1－2)

例） 一方通行の自動ドアの制御部

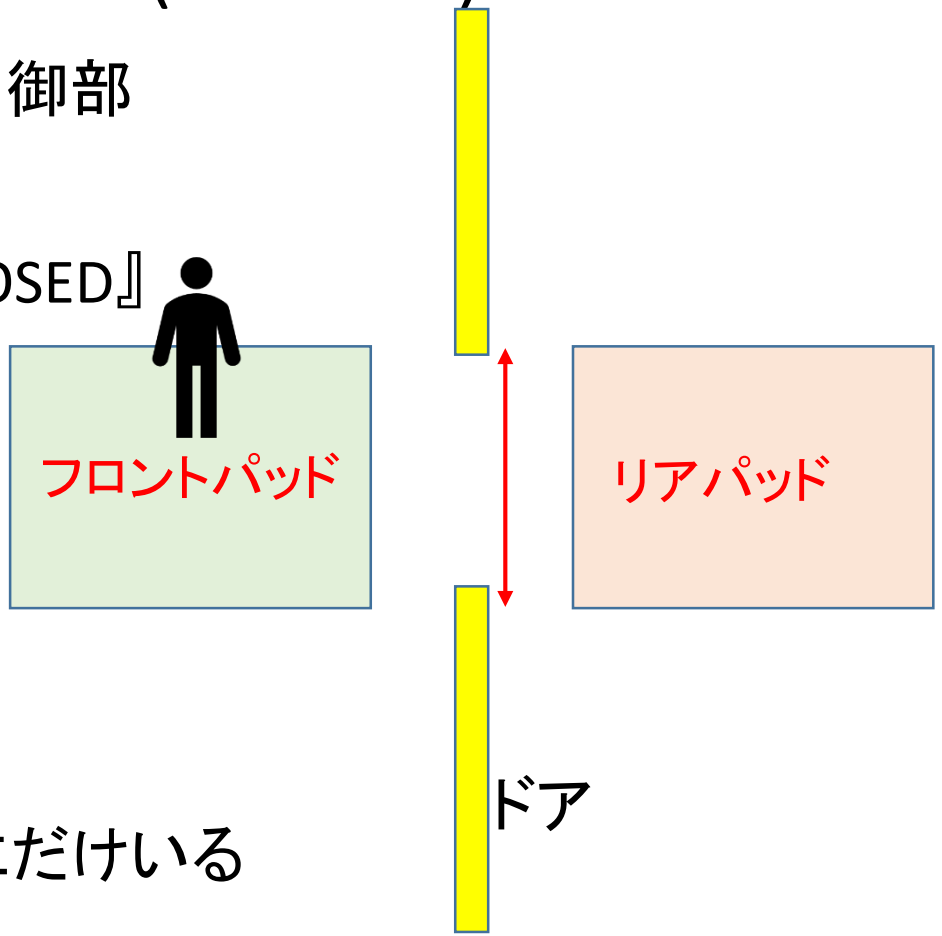
制御部の状態：『OPEN』か『CLOSED』



2.有限オートマトン (1－3)

例） 一方通行の自動ドアの制御部

制御部の状態:『OPEN』か『CLOSED』



入力:

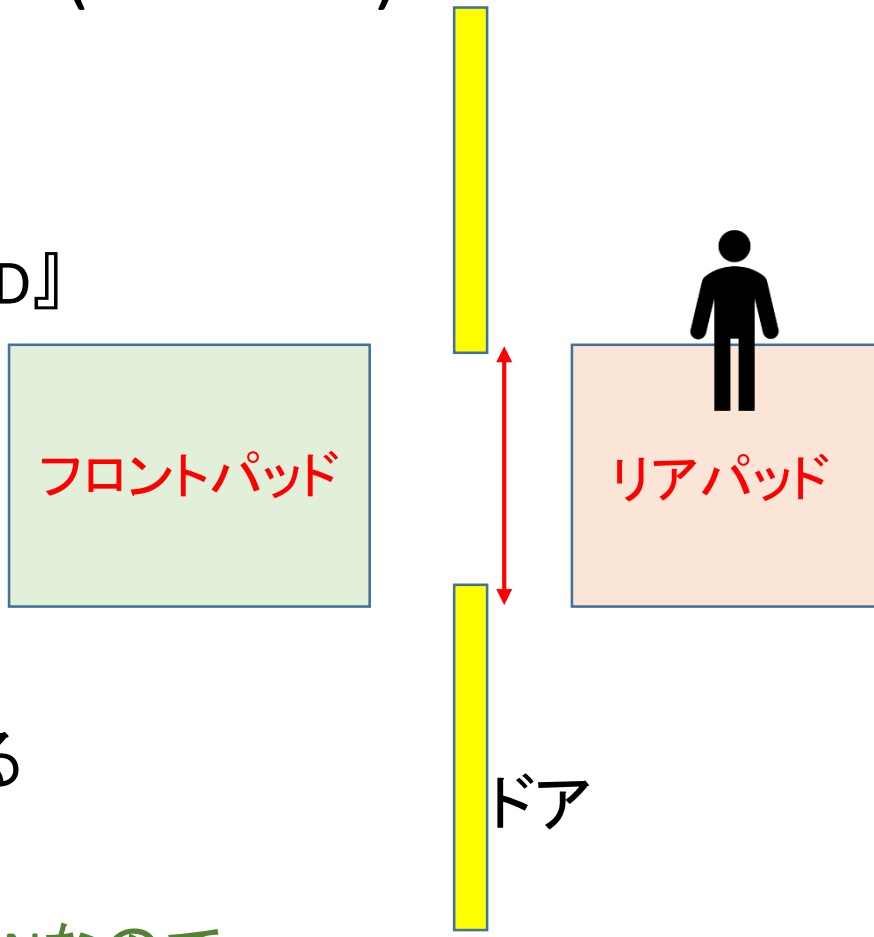
『FRONT』: 人がフロントパッドにだけいる

CLOSEDならOPENになる、OPENならOPENのまま

2.有限オートマトン (1－4)

例） 一方通行自動ドアの制御部

制御部の状態:『OPEN』か『CLOSED』



入力:

『REAR』: 人がリアパッドにだけいる

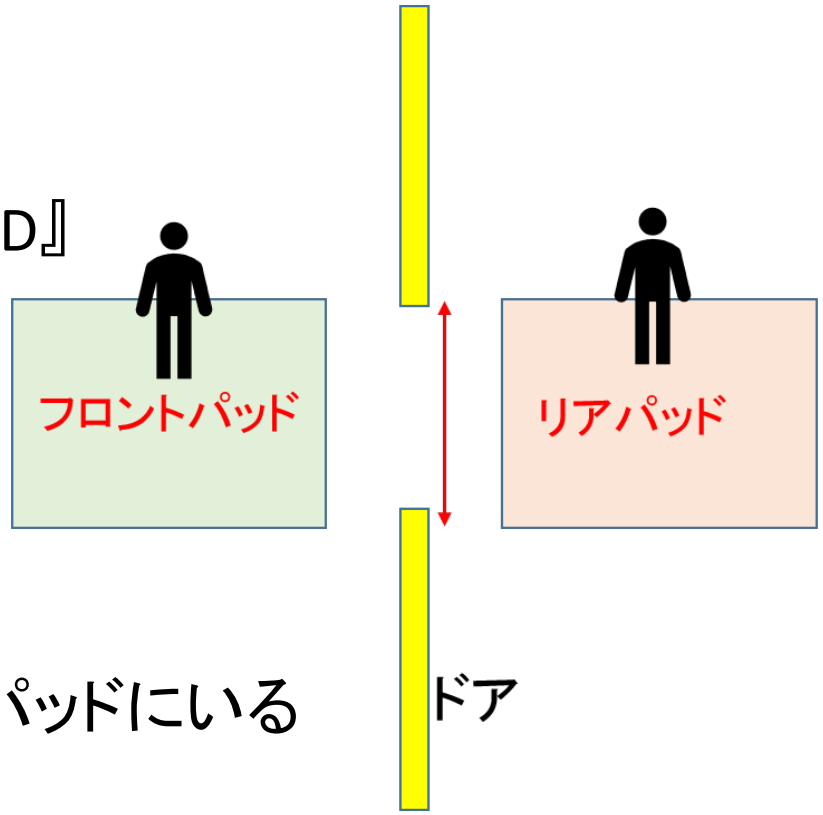
フロントからリアへ移動中ならOPENなので、

OPENのまま、逆にCLOSEDなら、リアからフロントに移動しようとしているので、CLOSEDのまま

2.有限オートマトン (1－5)

例） 一方通行自動ドアの制御部

制御部の状態:『OPEN』か『CLOSED』



入力:

『BOTH』: 人がフロントとリアの両パッドにいる

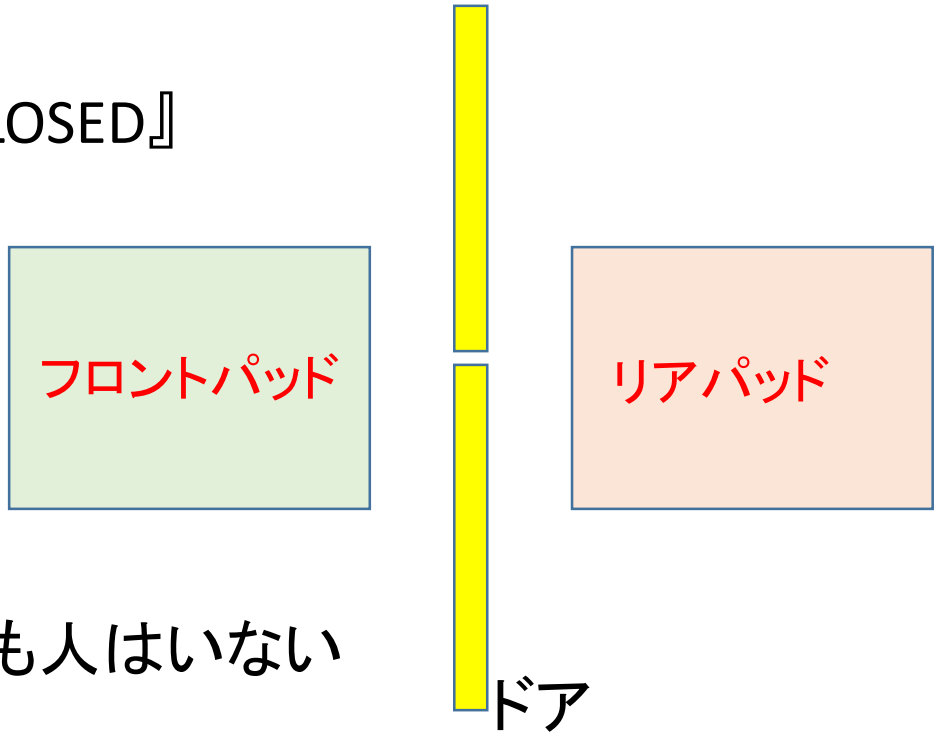
OPENならOPENのまま、

CLOSEDなら、OPENにする

2.有限オートマトン (1－6)

例） 一方通行自動ドアの制御部

制御部の状態:『OPEN』か『CLOSED』



入力:

『NEITHER』:どちらのパッドにも人はいない

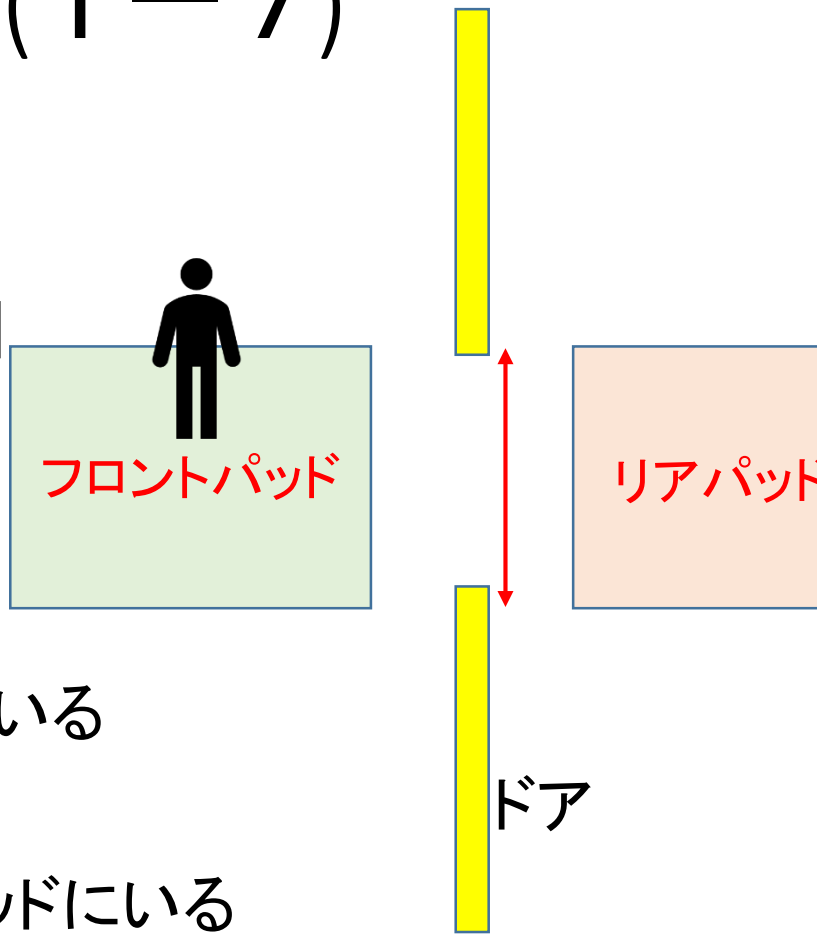
OPENならCLOSEDにする

CLOSEDならCLOSEDのまま

2.有限オートマトン (1－7)

例） 自動ドアの制御部

制御部の状態:『OPEN』か『CLOSED』



入力:

『FRONT』: 人がフロントパッドにだけいる

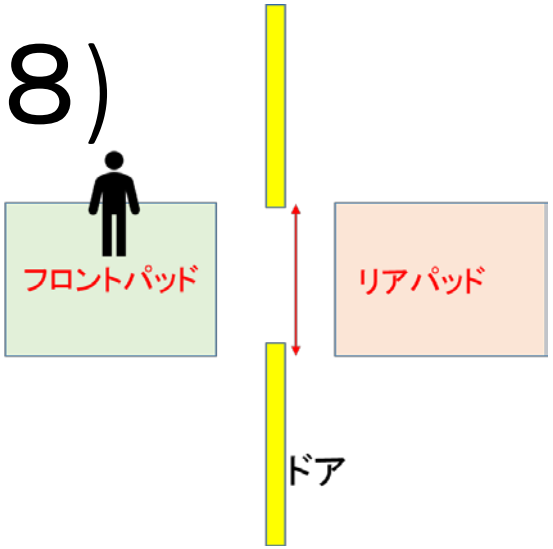
『REAR』: 人がリアパッドにだけいる

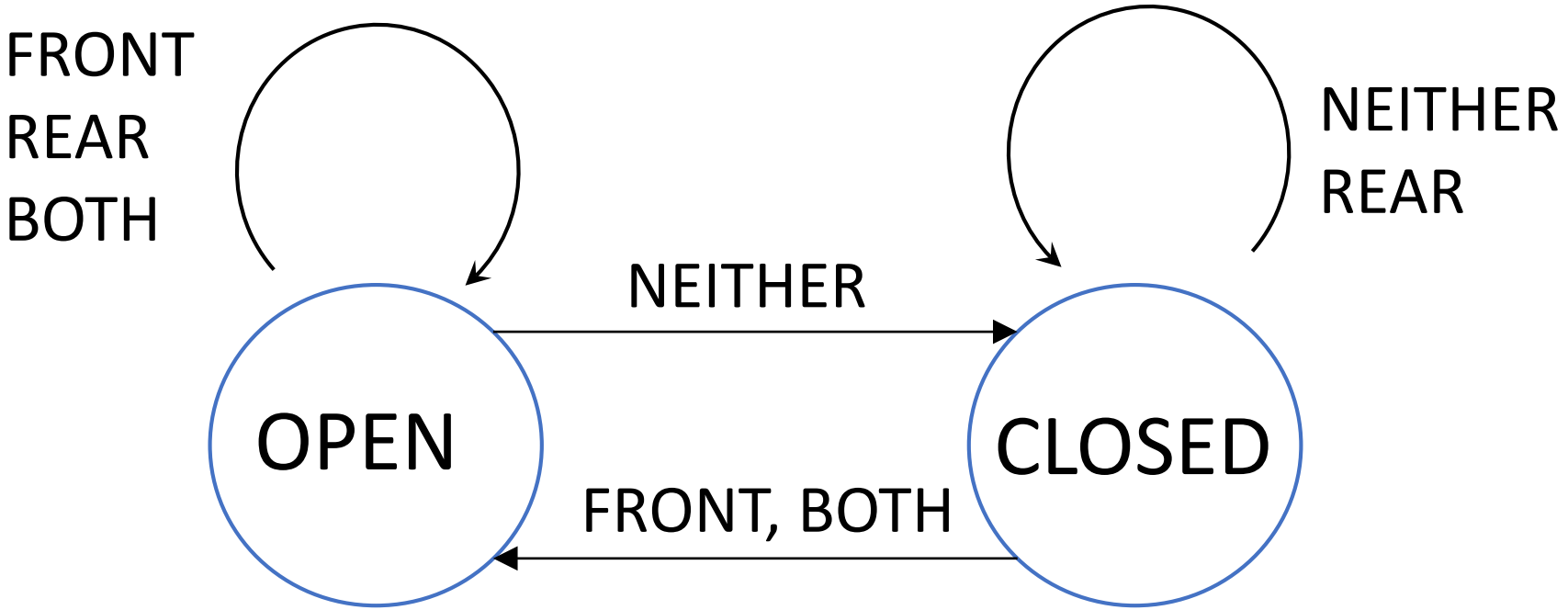
『BOTH』: 人がフロントとリアの両パッドにいる

『NEITHER』: どちらのパッドにも人はいない

2.有限オートマトン (1－8)

自動ドアの制御部の状態遷移図:



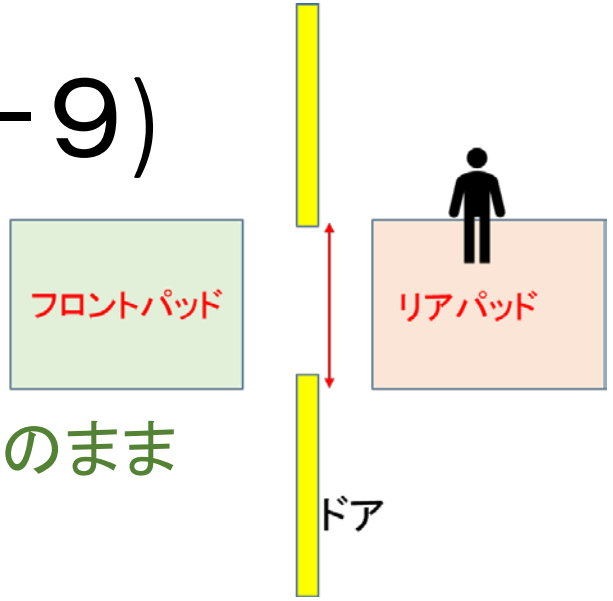


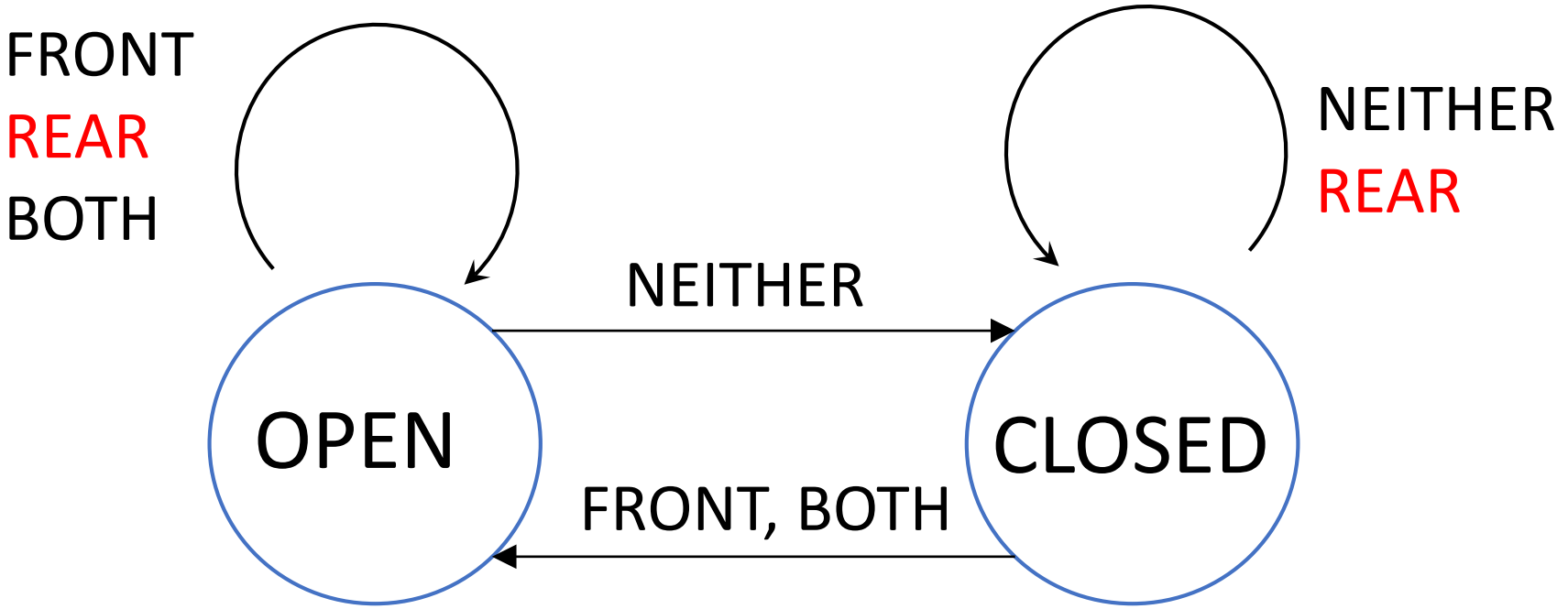
2.有限オートマトン(1－9)

自動ドアの制御部の状態遷移図:

入力:『REAR』:

OPENならOPENのまま、CLOSEDならCLOSEDのまま

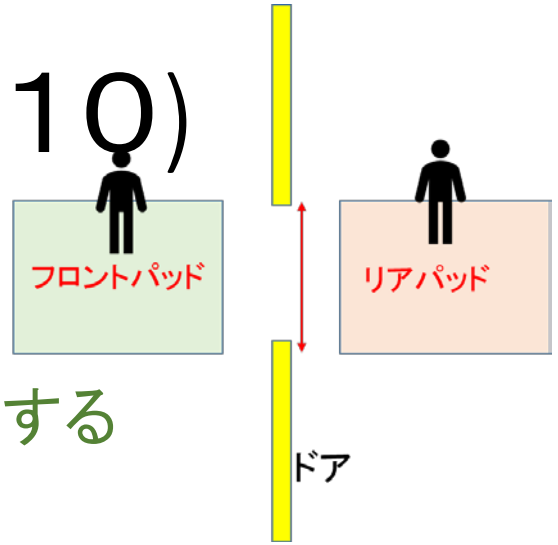


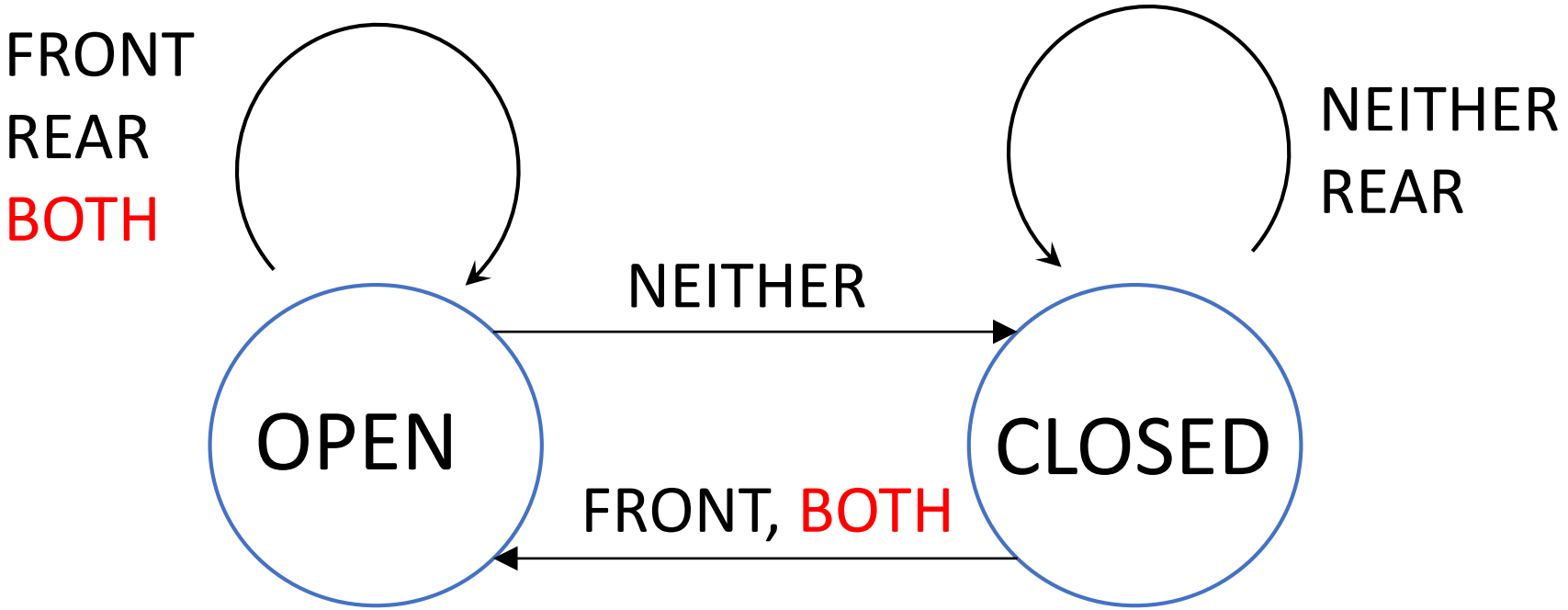


2.有限オートマトン(1－10)

自動ドアの制御部の状態遷移図:

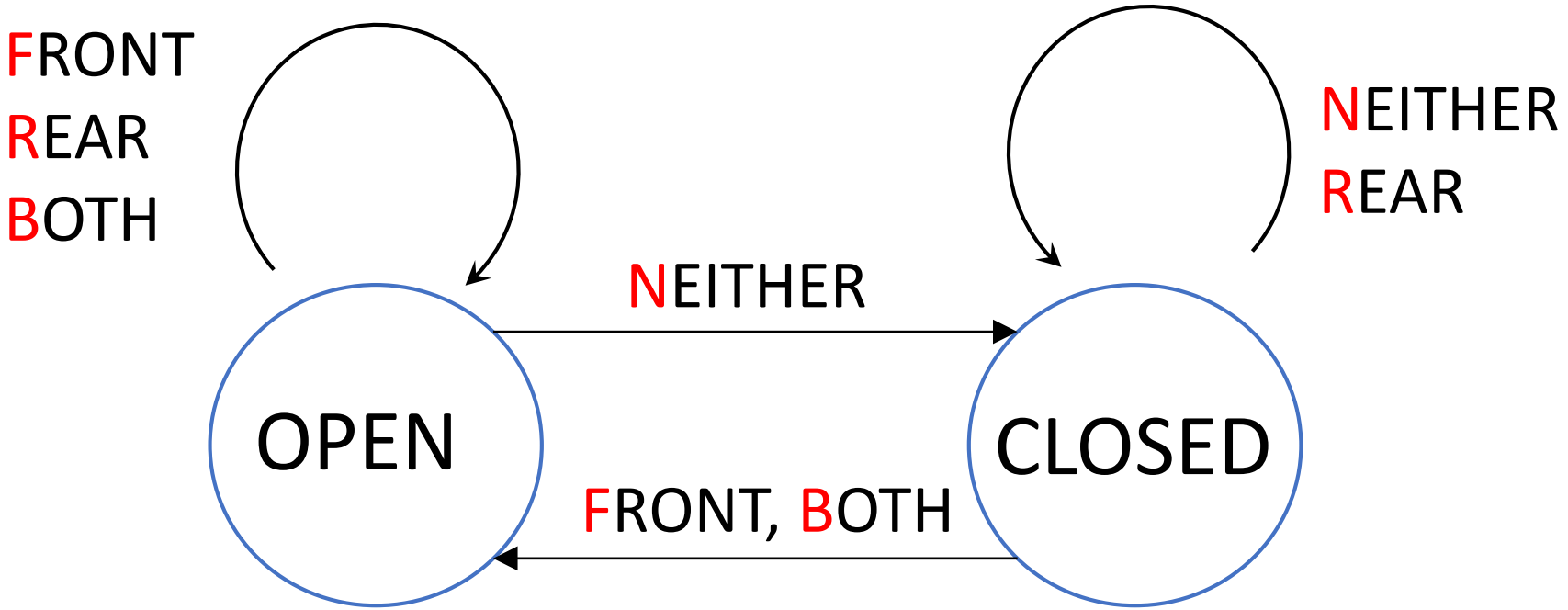
入力:『BOTH』:
OPENならOPENのまま、CLOSEDなら、OPENにする





2.有限オートマトン (1－11)

入力を頭文字のみ残して消す

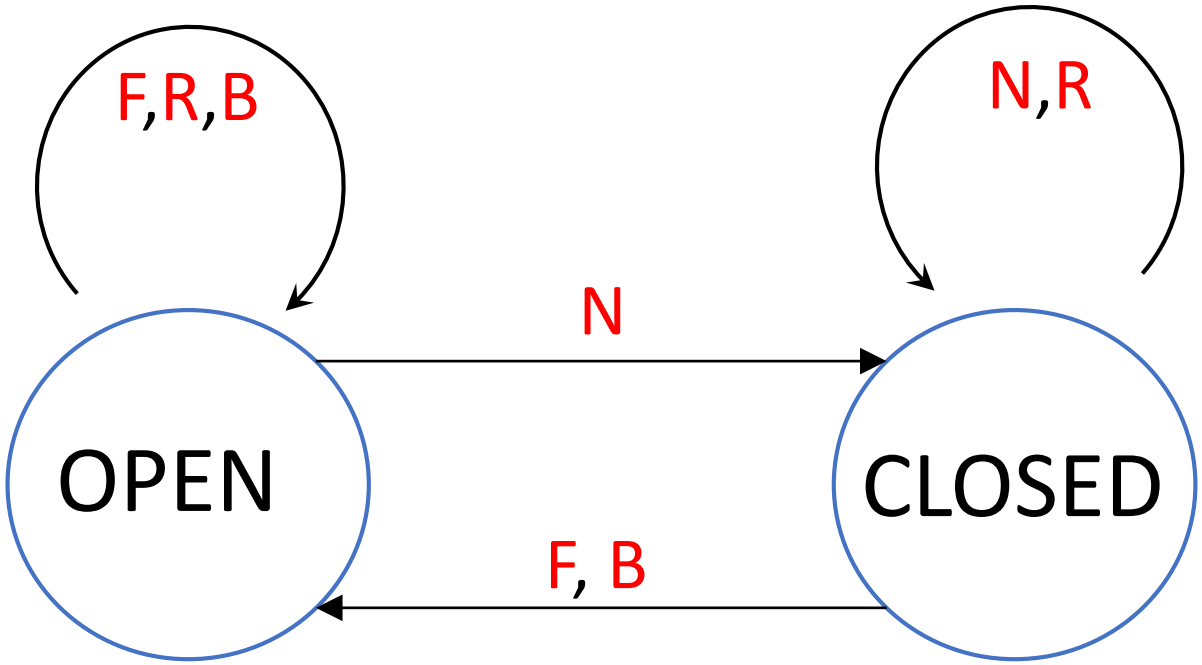


2.有限オートマトン (1－12)

4つの記号{F,R,B,N}を入力として受け取って状態を変える機械

今の状態と入力で、次の状態が決まる

⇔有限オートマトン



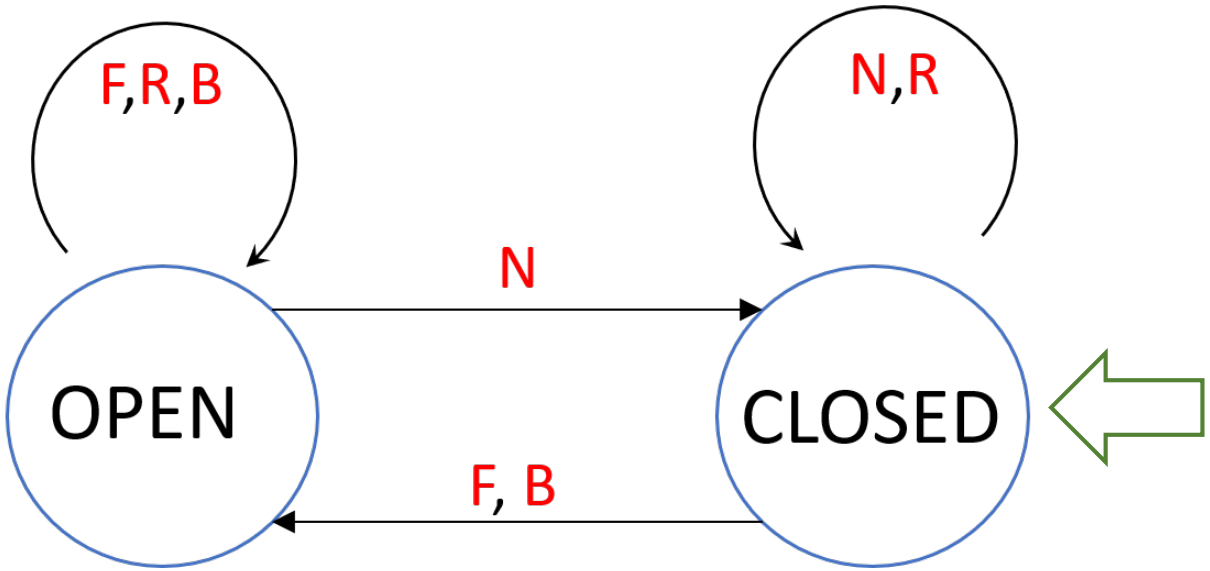
2.有限オートマトン (1－13)

最後の状態を出力と考える。

初期状態 : CLOSED

入力列 : “F B R N N R B”

出力 : OPEN

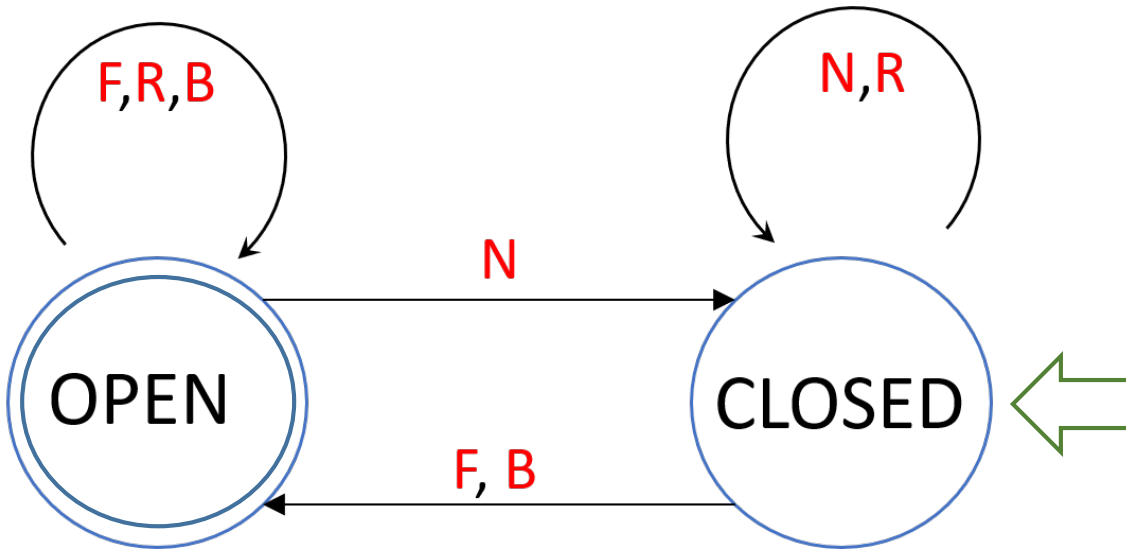


2.有限オートマトン (1－14)

初期状態 : CLOSED

入力列 : “F B R N N R B”

出力 : OPEN



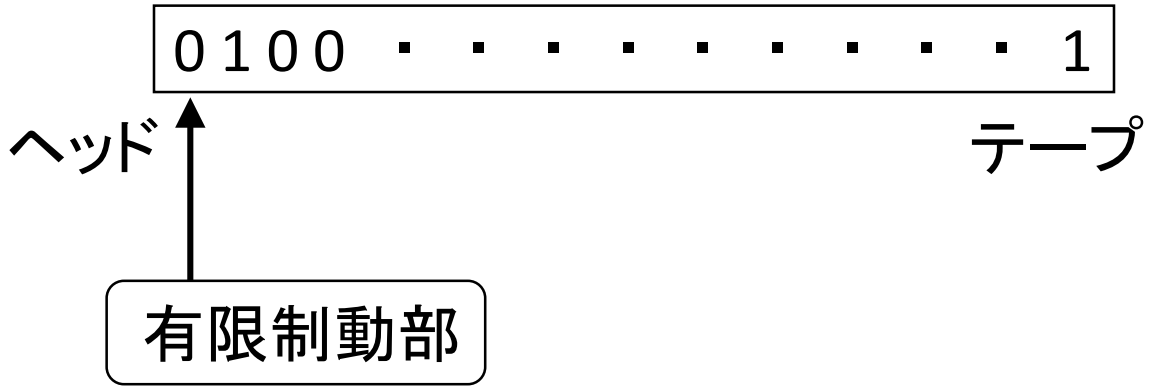
出力がOPENである入力列が『受理される』と考える。

『受理される』列の集合を言語Lとする： e.g. FBRNNRB$\in L$

Lはこのオートマトンで認識される言語と呼ばれる。

2.有限オートマトン (2)

有限オートマトンの概念図



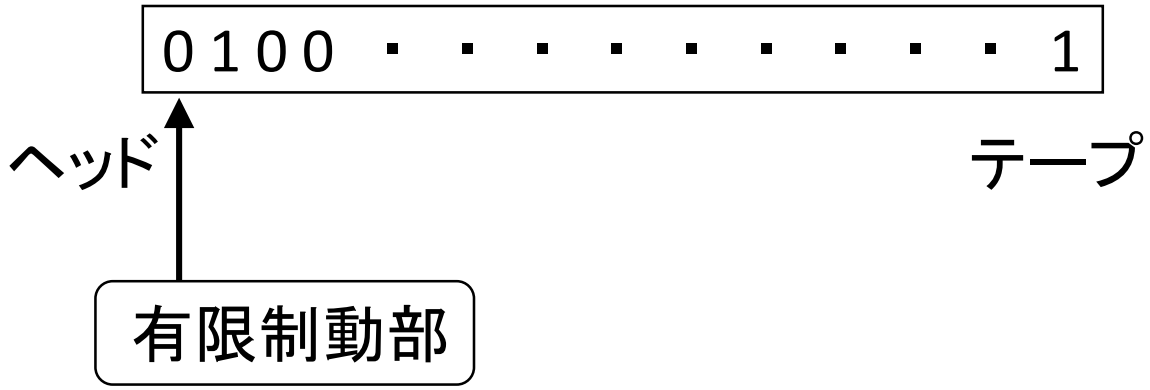
・テープと有限制動部からなる

・ヘッドがテープを左から右にスキャンする

・テープはマス目に分かれていて、各マスには記号が書いてある
　つまり、テープ全体で“列”になっている。

・テープをスキャンした後で、列が言語に属しているかどうかを判定する。

2.有限オートマトン (3)



定義: 有限オートマトン(fa)は$M = (K, \Sigma, \delta, s_0, F)$で与えられる。
K, Σ, F は有限集合

K : 状態の集合　（有限制動部の状態）
Σ : 入力記号の集合 （テープに書かれている記号）

$F \subseteq K$: 受理状態の集合 （状態がFで止まると入力を受理）

$s_0 \in K$: 初期状態 （状態は最初s_0である）
$\delta : K \times \Sigma \to K$ 状態遷移関数
（$s' = \delta(s, a)$なら、状態sで記号aを読むと状態はs'に変わる）

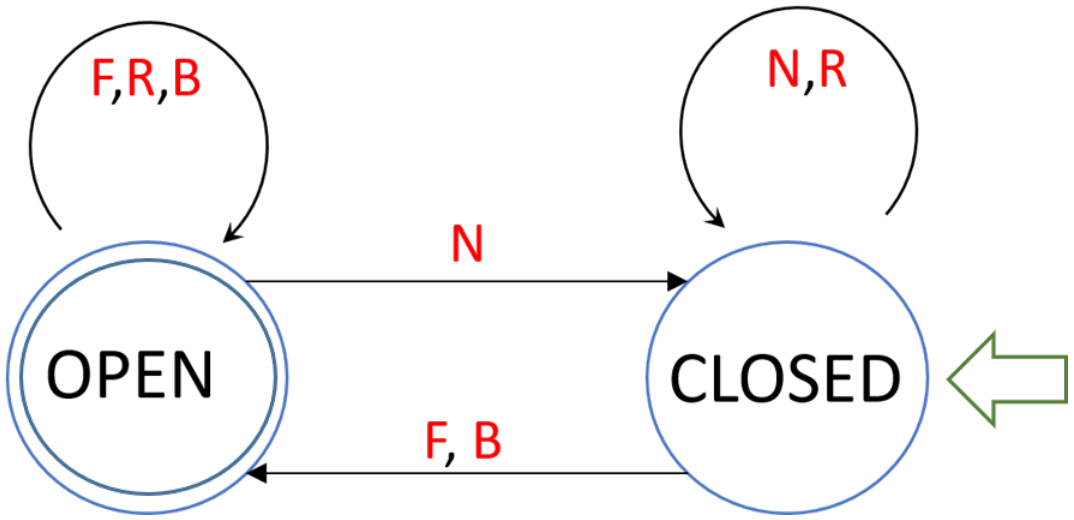
2.有限オートマトン (4)

定義: 有限オートマトン(fa)は$M = (K, \Sigma, \delta, s_0, F)$で与えられる。
K : 状態の集合、　Σ : 入力記号の集合

$F \subseteq K$: 受理状態の集合、 $s_0 \in K$: 初期状態
$\delta : K \times \Sigma \to K$ 状態遷移関数

(例) $K = \{\mathrm{OPEN}, \mathrm{CLOSED}\}, \Sigma = \{\mathrm{F}, \mathrm{R}, \mathrm{B}, \mathrm{N}\}, F = \{\mathrm{OPEN}\}$

初期状態$s_0 = \mathrm{CLOSED}$



$\delta(\mathrm{OPEN}, \mathrm{F}) = \mathrm{OPEN},$
$\delta(\mathrm{OPEN}, \mathrm{N}) = \mathrm{CLOSED},$
$\delta(\mathrm{CLOSED}, \mathrm{F}) = \mathrm{OPEN}\ ,$
$\delta(\mathrm{CLOSED}, \mathrm{N}) = \mathrm{CLOSED},$

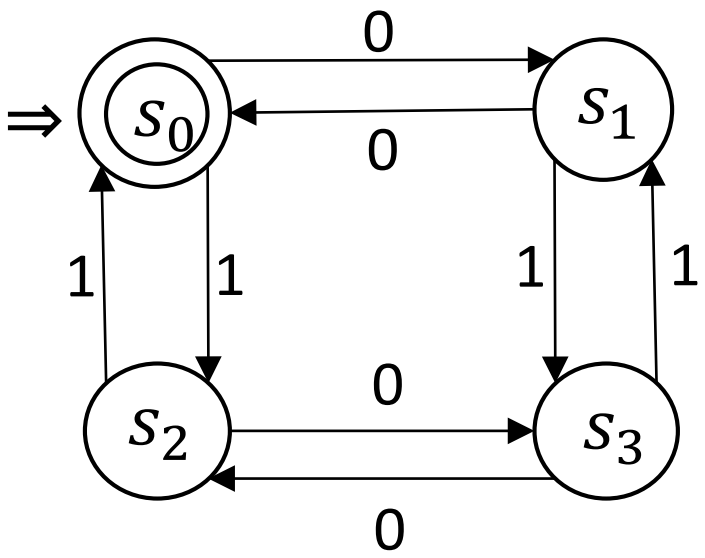
… etc

2.有限オートマトン (5)

定義: 有限オートマトン(fa)は$M = (K, \Sigma, \delta, s_0, F)$で与えられる。
K : 状態の集合、　Σ : 入力記号の集合

$F \subseteq K$: 受理状態の集合、 $s_0 \in K$: 初期状態
$\delta : K \times \Sigma \to K$ 状態遷移関数

(例) $K = \{s_0, s_1, s_2, s_3\}, \Sigma = \{0,1\}, F = \{s_0\}$



状態遷移図

・状態は丸で示す

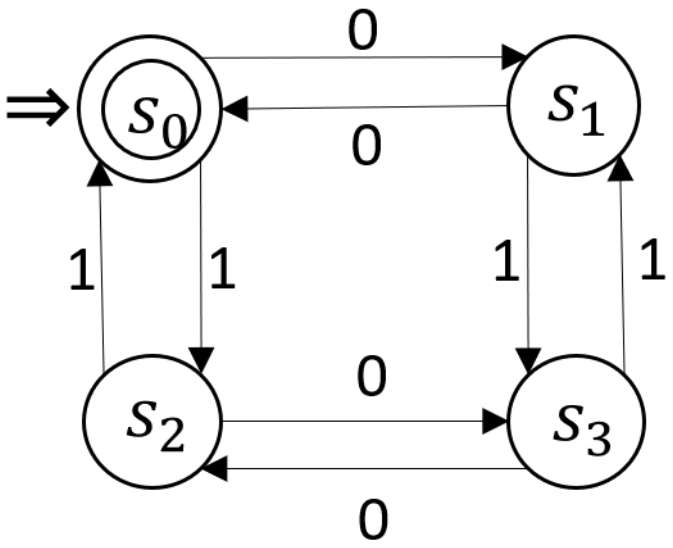
・初期状態を”⇒”で示す

・受理状態は二重丸

・状態遷移は矢の形で示す

e.g. $\delta(s_0, 0) = s_1, \delta(s_0, 1) = s_2$

2.有限オートマトン (6)



110をこの順で読むと、$s_0 \to s_2 \to s_0 \to s_1$ となる。
写像δを$\hat{\delta}(s_0, 110) = s_1$となるように(再帰的)定義をする:

定義$(\hat{\delta}: K \times \Sigma^* \to K)$

$\forall s \in K, \forall a \in \Sigma, \forall x \in \Sigma^*$ に対して、

$$\hat{\delta}(s, \epsilon) = s \text{ かつ } \hat{\delta}(s, xa) = \delta\left(\hat{\delta}(s, x), a\right)$$

2.有限オートマトン (7)

定義 $(\hat{\delta}: K \times \Sigma^* \to K)$

$\forall s \in K, \forall a \in \Sigma, \forall x \in \Sigma^*$ に対して、

$$\hat{\delta}(s, \epsilon) = s \text{ かつ } \hat{\delta}(s, xa) = \delta\left(\hat{\delta}(s, x), a\right)$$

(例)

110をこの順で読むと、$s_0 \to s_2 \to s_0 \to s_1$

$\hat{\delta}(s_0, \epsilon) = s_0,$

$\hat{\delta}(s_0, 1) = \hat{\delta}(s_0, \epsilon 1) = \delta\left(\hat{\delta}(s_0, \epsilon), 1\right) = \delta(s_0, 1) = s_2,$

$\hat{\delta}(s_0, 11) = \delta\left(\hat{\delta}(s_0, 1), 1\right) = \delta(s_2, 1) = s_0$

$\hat{\delta}(s_0, 110) = \delta\left(\hat{\delta}(s_0, 11), 0\right) = \delta(s_0, 0) = s_1$

2.有限オートマトン (8)

定義 $(\hat{\delta}: K \times \Sigma^* \to K)$

$\forall s \in K, \forall a \in \Sigma, \forall x \in \Sigma^*$ に対して、

$$\hat{\delta}(s, \epsilon) = s \text{ かつ } \hat{\delta}(s, xa) = \delta\left(\hat{\delta}(s_0, x), a\right)$$

記号$a \in \Sigma$に対して、$\delta(s, a) = \hat{\delta}(s, a)$なので、

ハット"^"を省略しても混乱が生じない。

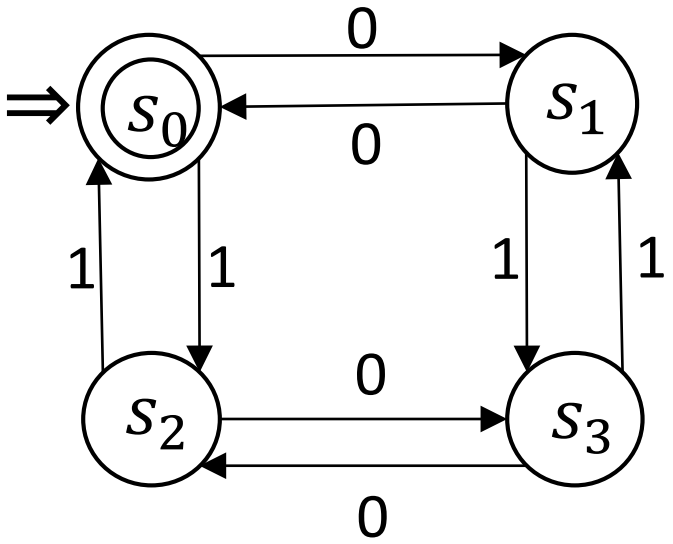
結局、δの定義域が$K \times \Sigma$から$K \times \Sigma^*$に自然に拡張された。

定義 fa $M = (K, \Sigma, \delta, s_0, F)$と$x \in \Sigma^*$に対して、

$\delta(s_0, x) \in F$のとき、xはMに受理されると呼ぶ。
Mが受理する列の全体をMが認識する言語と呼ぶ。

2.有限オートマトン (9)

fa M_1の状態遷移図



例題2.4　M_1の認識する言語を求めよ。

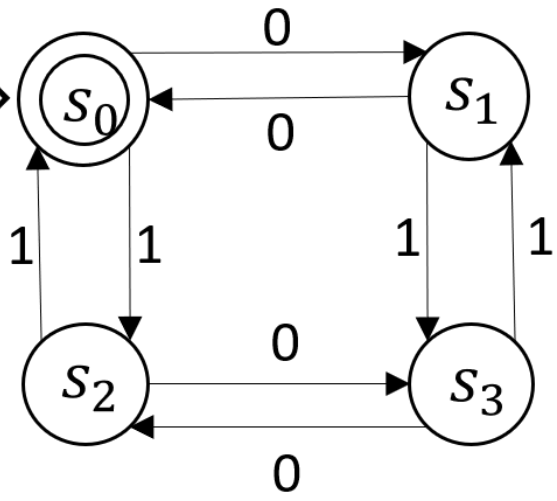
（解答）

0の個数と1の個数がともに偶数となる$\{0,1\}$上の列となる。

「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、$\#_0(x)$と$\#_1(x)$はともに偶数。
$s = s_1$なら、$\#_0(x)$が奇数、$\#_1(x)$は偶数、
$s = s_2$なら、$\#_0(x)$が偶数、$\#_1(x)$が奇数、
$s = s_3$なら、$\#_0(x)$と$\#_1(x)$はともに奇数」であることを証明する。

2.有限オートマトン(10)

「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、$\#_0(x)$と$\#_1(x)$はともに偶数。
$s = s_1$なら、$\#_0(x)$が奇数、$\#_1(x)$は偶数、
$s = s_2$なら、$\#_0(x)$が偶数、$\#_1(x)$が奇数、
$s = s_3$なら、$\#_0(x)$と$\#_1(x)$はともに奇数」



（証明）

i) 列xの長さが0のとき、定義より$\delta(s_0, \epsilon) = s_0$となりOK

ii) 列xの長さがnのとき成立していると仮定する。

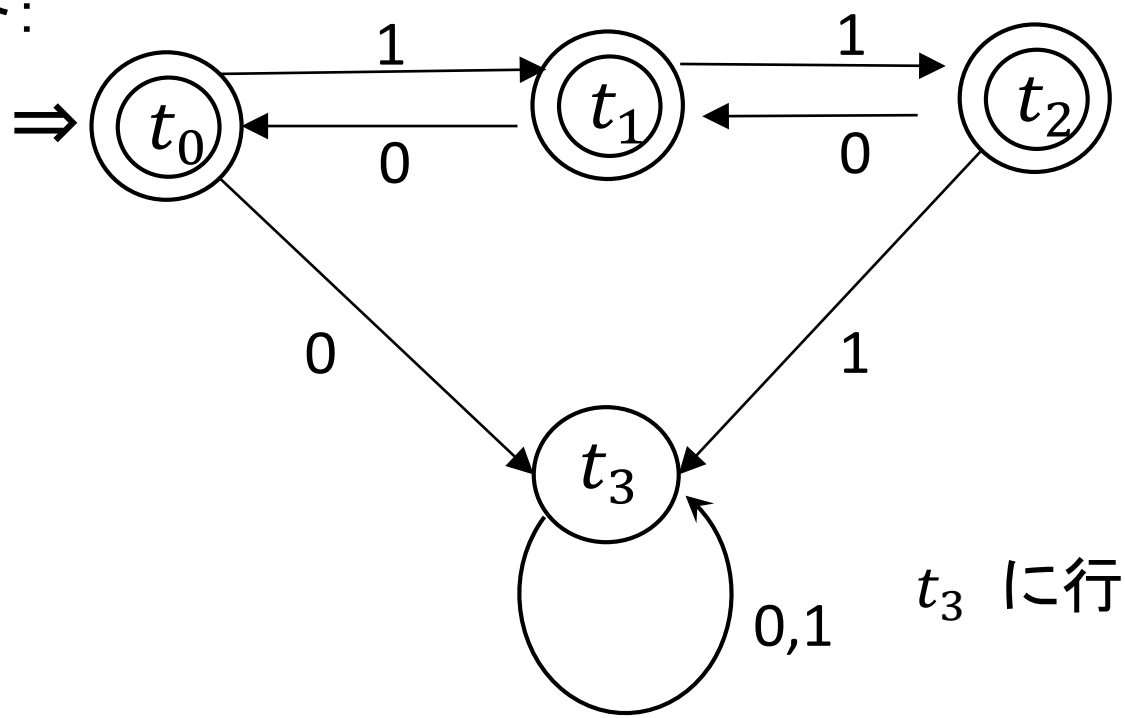
長さ$n+1$の列yは$y = xa$と書ける。
$\#_0(x)$が偶数、$\#_1(x)$が奇数で、$a = 0$のときを考える。
仮定よりxを読み終わった後の状態はs_2.
状態遷移図より$a = 0$を読むと$s_2 \to s_3$となる。
yは$\#_0(x)$と$\#_1(x)$がともに奇数なのでOK。
他もすべて調べると証明が終わる。（以下省略)

2.有限オートマトン(11)

例題　言語$L_1 := \{x \mid x \in \{0,1\}^*$かつ$x$の任意のプレフィックス$y$
に対して、$0 \leq \#_1(y) - \#_0(y) \leq 2\}$

を認識するfaを与えよ。

解答:



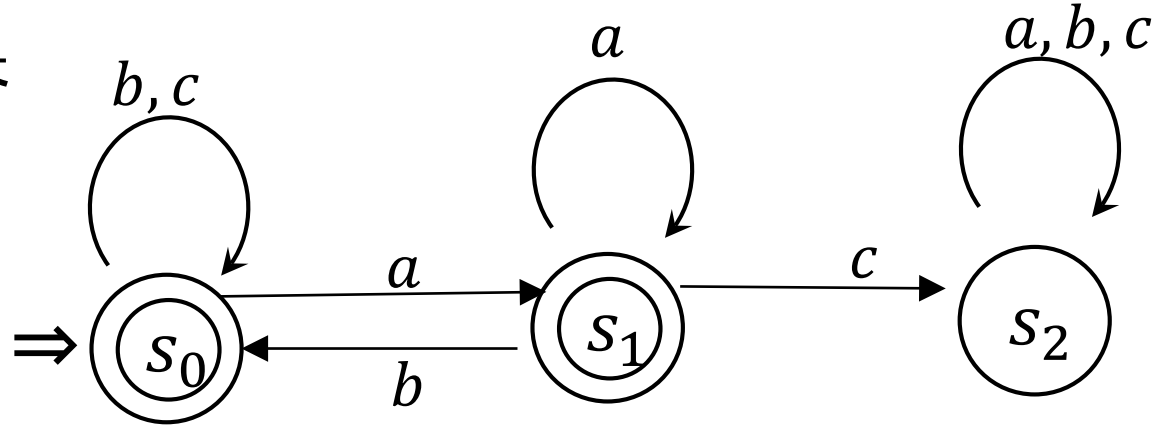
t_3 に行くとかえって来れない。

2.有限オートマトン(12)

例題

言語$L_2 := \{x \mid x \in \{a,b,c\}^*$かつ$x$は部分列$ac$を含まない$\}$
を認識するfaを与えよ。

解答



s_1は、直前にaを読んだことを記憶するための状態。

s_0とs_1でaを読むとs_1に行く。その次にcを読むとs_2に行き受理されなくなる。

問題

以下の言語に認識するdfaの状態遷移図を書け。ただし、アルファベットは$\{0,1\}$とする。
(つまり$w \in \{0,1\}^*$):

① $\{w \mid w$は1で始まり0で終わる$\}$

② $\{w \mid w$は少なくとも三つの1を含む$\}$

③ $\{w \mid w$は部分列0101を含む$\}$

④ $\{w \mid 3 \leq |w|$ かつ wの3番目の文字が0 $\}$

⑤ $\{w \mid w$は 0で始まり長さが奇数、
もしくは、1で始まり長さが偶数$\}$

⑥ $\{w \mid w$は部分列110を含まない$\}$

⑦ $\{w \mid |w| \leq 5\}$

問題(前ページの続き)

⑧ $\{w \mid w \neq 11$ かつ $w \neq 111\}$

⑨ $\{w \mid w$の全ての奇数番目は$1\}$

⑩ $\{w \mid \#_1(w) \geq 2$ かつ$\#_0(w) \geq 1\}$

⑪ $\{0, \epsilon\}$

⑫ $\{w \mid \#_0(w)$が偶数、もしくは$\#_1(w) = 2\}$

⑬ $\emptyset$

⑭ $\{w \mid w \neq \epsilon\}$

⑮ $\{w \mid |w|$が偶数、もしくは$|w|$が3の倍数$\}$

2.有限オートマトン(13)

直積オートマトン(直積fa)

入力アルファベットが等しい二つのfa

$M_1 = (K_1, \Sigma, \delta_1, s_{01}, F_1)$と$M_2 = (K_2, \Sigma, \delta_2, s_{02}, F_2)$に対して、

状態:$(s_1, s_2) \in K_1 \times K_2$

状態遷移関数:$\delta((s_1, s_2), a) \mapsto (\delta_1(s_1, a), \delta_2(s_2, a))$

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たものを指定する。

2.有限オートマトン (14)

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

$M_1 = (K_1, \Sigma, \delta_1, s_{01}, F_1)$と$M_2 = (K_2, \Sigma, \delta_2, s_{02}, F_2)$ とする。
それらの直積faを以下の初期状態と受理状態集合で作る:

初期状態 : (s_{01}, s_{02})
受理状態集合 : $F := \{(s_1, s_2) \mid s_1 \in F_1, s_2 \in F_2\}$

$x \in L_1 \cap L_2$なら、$\delta_1(s_{01}, x) \in F_1$かつ$\delta_2(s_{02}, x) \in F_2$.

$\Longrightarrow \ \delta\big((s_{01}, s_{02}), x\big) = \big(\delta_1(s_{01}, x), \delta_2(s_{02}, x)\big) \in F$

$\Longrightarrow$ 直積オートマトンはxを受理

2.有限オートマトン (15)

$x \notin L_1 \cap L_2$なら、$\delta_1(s_{01}, x) \notin F_1$もしくは$\delta_2(s_{02}, x) \notin F_2$.

$\Longrightarrow \quad \delta\big((s_{01}, s_{02}), x\big) = \big(\delta_1(s_{01}, x), \delta_2(s_{02}, x)\big) \notin F$

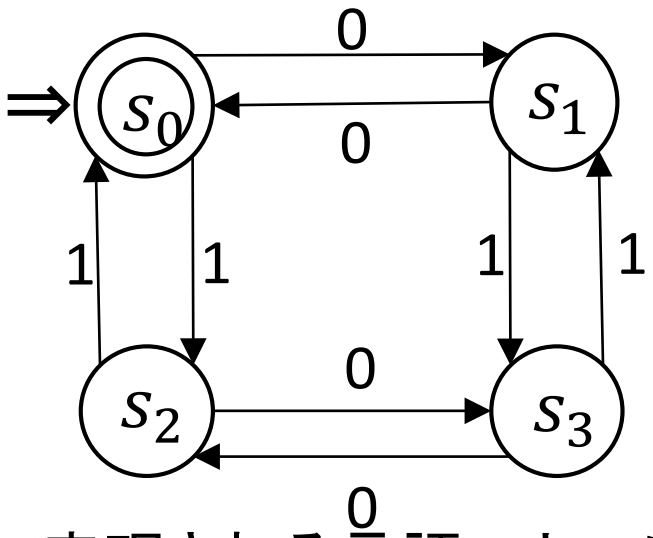
$\Longrightarrow$ 直積オートマトンはxを受理しない。

よって、直積オートマトンは、$L_1 \cap L_2$を認識する。 ■

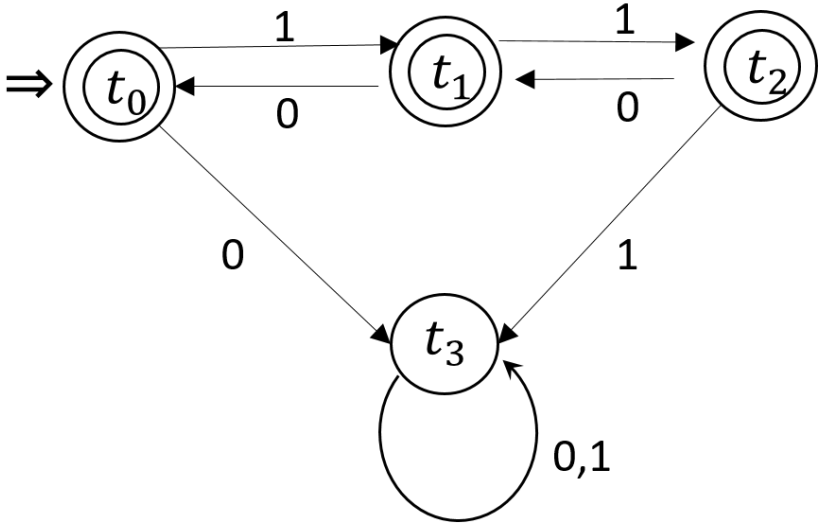
2.有限オートマトン(16)

直積オートマトンの例:

以前の2つのオートマトン



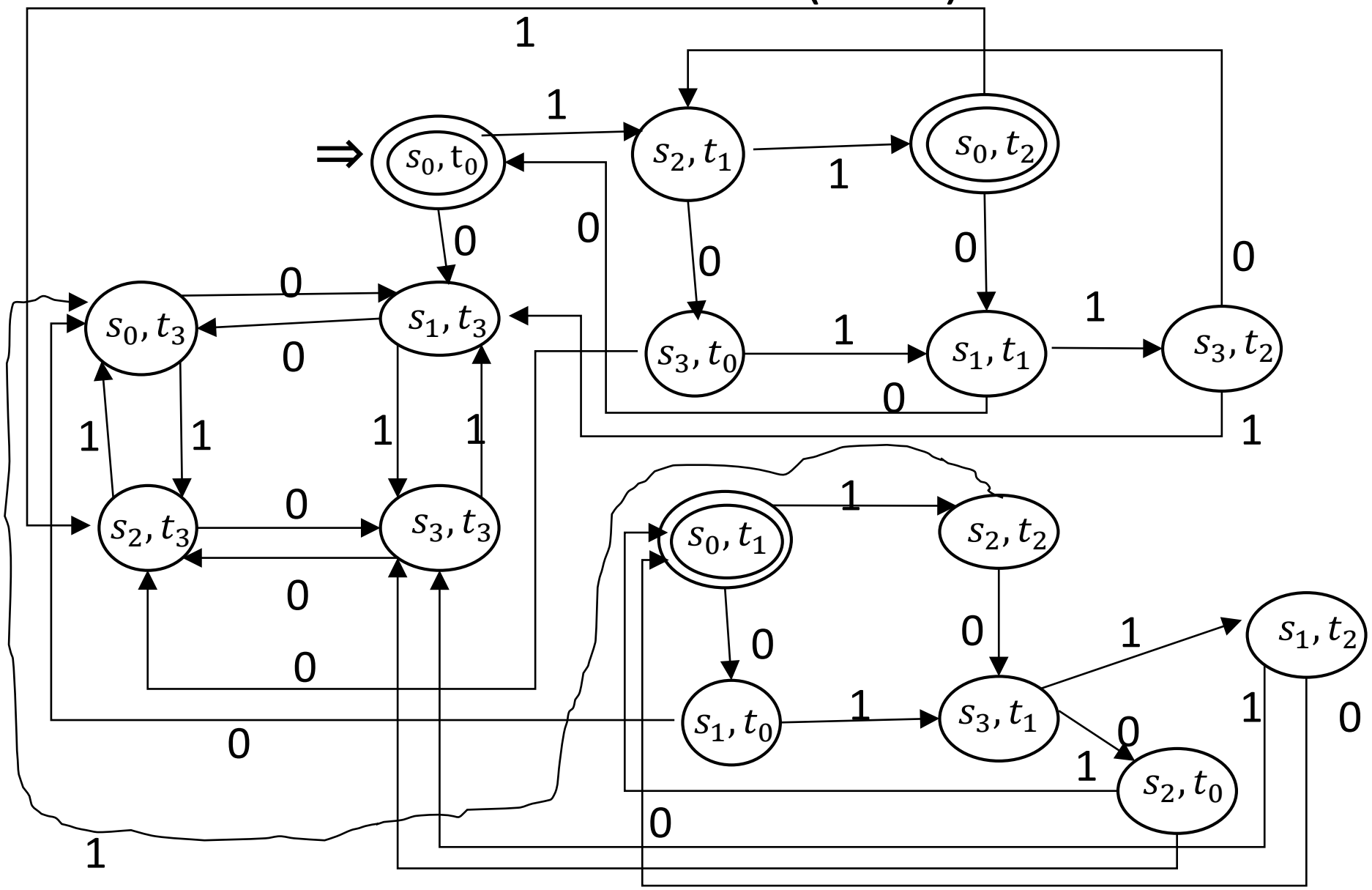
と



で表現される言語L_1とL_2に対して、
$L_1 \cap L_2$を認識するオートマトンを直積オートマトンで与える:

状態の数は、$(s_0, t_0), \cdots, (s_3, t_3)$の16個
初期状態は (s_0, t_0)、受理状態は$(s_0, t_0), (s_0, t_1), (s_0, t_2)$

2.有限オートマトン (17)



2.有限オートマトン(18)

faの記憶容量　＝　状態の個数

同じ言語を認識する限り、より状態の少ないfaが効率の良いfa.

初期状態から到達できない状態は排除しても認識する言語は変わらない。(冗長な状態)

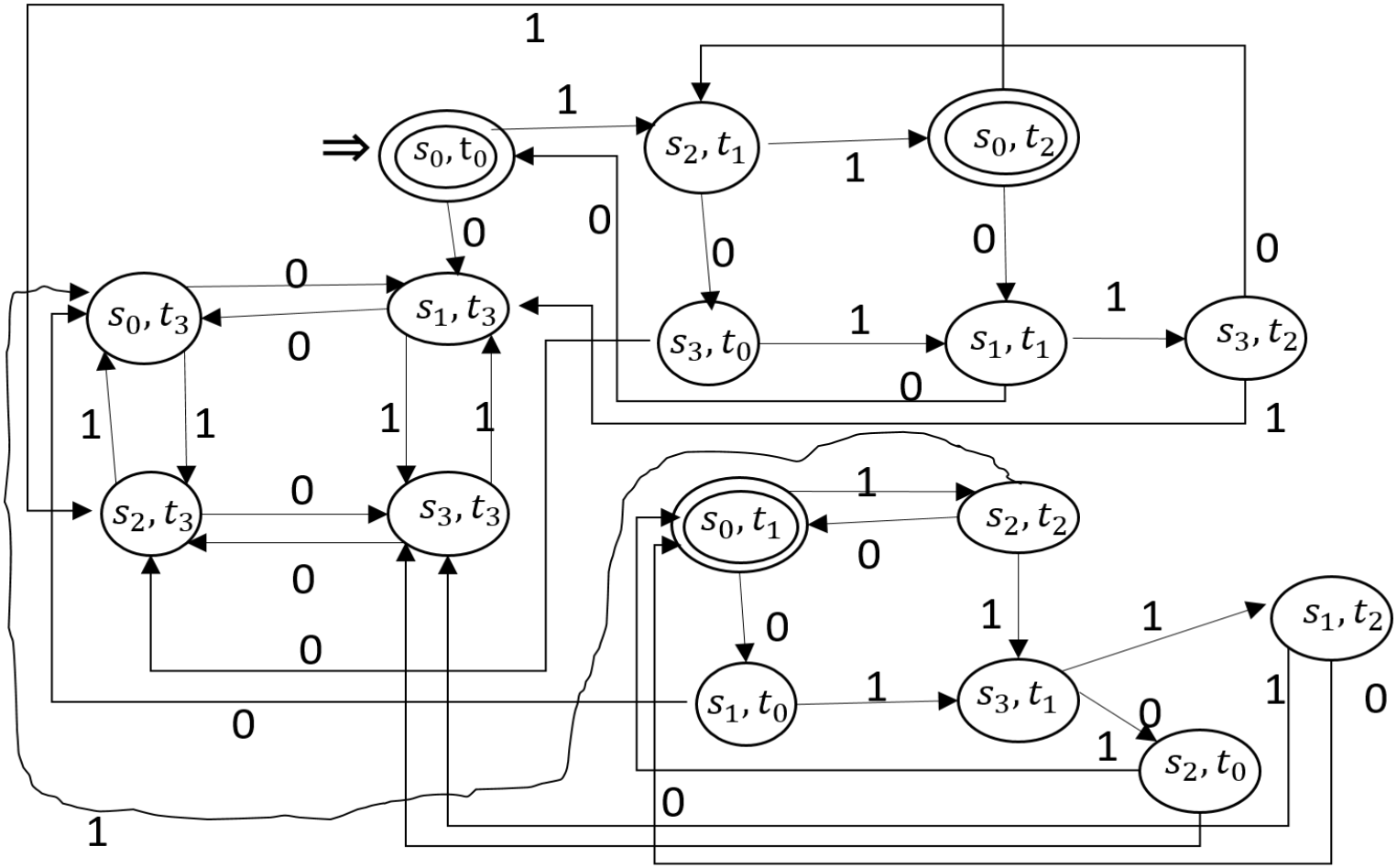
先ほどの直積オートマトンの例では、

(s_0, t_1)は初期状態から到達できないので、

削除しても認識する言語は変わらない。

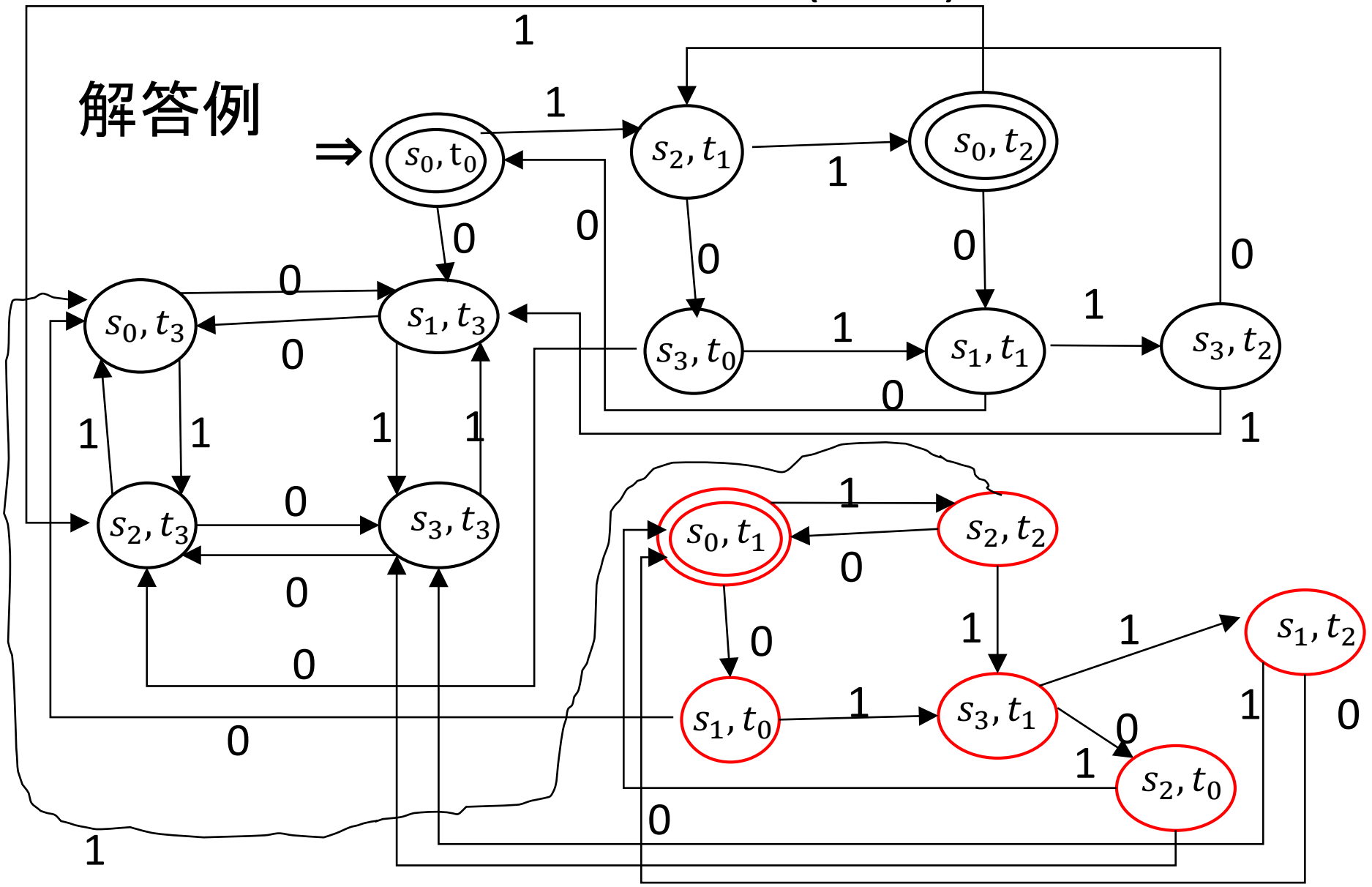
2.有限オートマトン(19)

練習問題：(s_0, t_1)以外にも、初期状態から到達できない状態が5つ存在するので探してみよ。

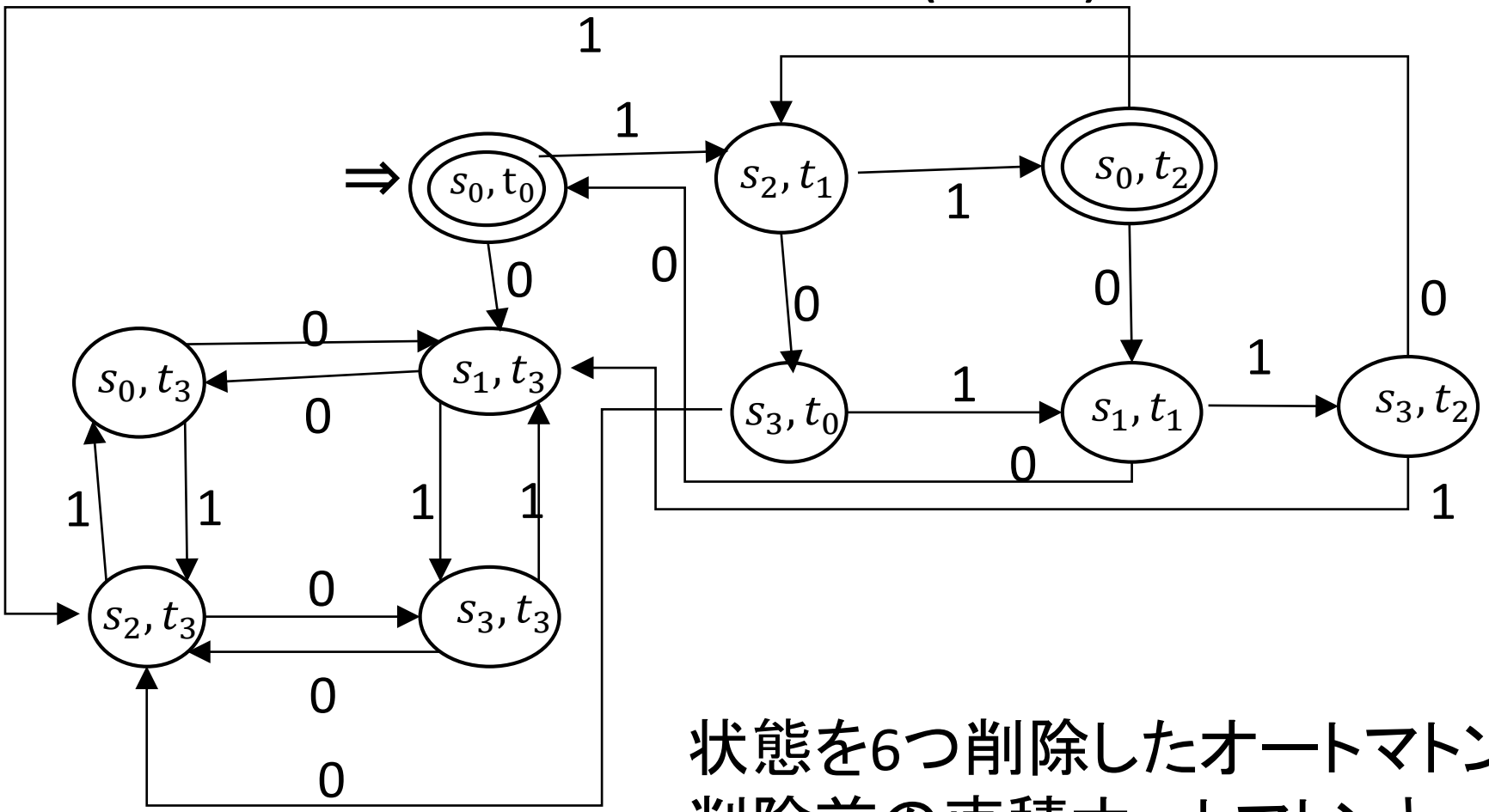


2.有限オートマトン (20)

解答例



2.有限オートマトン (21)



状態を6つ削除したオートマトン
削除前の直積オートマトンと
同じ言語を認識する。

2.有限オートマトン (22)

今後、faは初期状態から到達できない状態は含まないと仮定する。

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。

定義（状態の等価性）

fa $M = (K, \Sigma, \delta, s_0, F)$の二つの状態$s_1$と$s_2$は、
『全ての列$x \in \Sigma^*$に対して、
$\delta(s_1, x) \in F \iff \delta(s_2, x) \in F$』
を満たすとき等価であるという。

2.有限オートマトン (23)

定理 2.3

(i) fa Mが等価な2個の状態を有するならば、Mと同じ言語を認識して状態数がMより1個少ないfa M'が存在する。

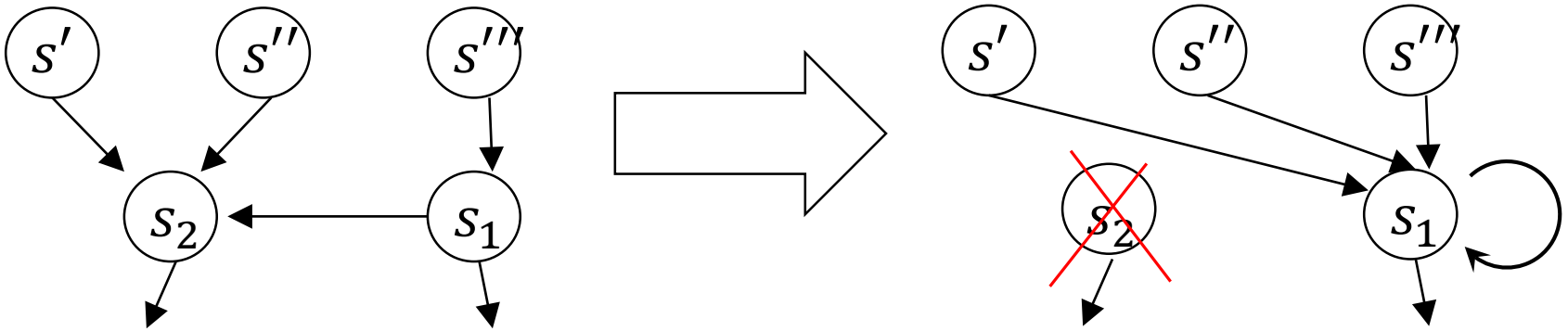
(ii) fa M には、初期状態から到達できない状態は存在しないとする。このとき、Mと同じ言語を認識して状態数がより少ないfa M'が存在するならば、Mには(少なくとも)2個の等価な状態が存在する。

2.有限オートマトン (24)

(i) fa Mが等価な2個の状態を有するならば、Mと同じ言語を認識して状態数がMより1個少ないfa M'が存在する。

(証明の概要)

s_1とs_2が等価だとする。
状態sからs_2への遷移が存在するなら、それらを全てsからs_1への遷移に変更する。すると、s_2への遷移がなくなるので、s_2を削除したfaをM'とすると、M'はMより状態が一つ少ない。MとM'の等価性も証明できる。



2.有限オートマトン (25)

(ii) fa Mには、初期状態から到達できない状態は存在しないとする。このとき、Mと同じ言語を認識して状態数がより少ないfa M'が存在するならば、Mには必ず2個の等価な状態が存在する。

証明は複雑なので省略する。(知りたい人は参考書)

2.有限オートマトン (26)

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faで認識できる。

(証明)　Lを受理するfaをMとする。
『Mが受理しない列』を受理するfaを作りたい。

Mが列xを受理しない ⇔　列xを読み終わると非受理状態にある

よって、Mの受理状態と非受理状態を入れ替えたfaをM'とすると、
M'は$\Sigma^* - L$を認識する。

問題

以下の言語に認識するdfaの状態遷移図を書け。ただし、アルファベットは$\{a, b\}$とする($w \in \{a, b\}^*$)。

① $\{w \mid \#_a(w) \geq 3, \text{かつ}, \#_b(w) \geq 2\}$

② $\{w \mid \#_a(w) = 2, \text{かつ}, \#_b(w) \geq 2\}$

③ $\{w \mid \#_a(w)\text{は偶数}, \text{かつ}, \#_b(w) \in \{1,2\}\}$

④ $\{w \mid \#_a(w)\text{は偶数}, \text{かつ}, \text{各々の}a\text{は}b\text{の直後にある}\}$

① $\{w \mid w\text{は}\ a\text{で始まり}, \text{かつ}, \#_b(w) \leq 1\}$

② $\{w \mid \#_a(w)\text{は奇数}, \text{かつ}, \ w\text{は}b\text{で終わる}\}$

③ $\{w \mid |w|\text{は偶数}, \text{かつ}, \#_a(w)\text{は奇数}, \}$

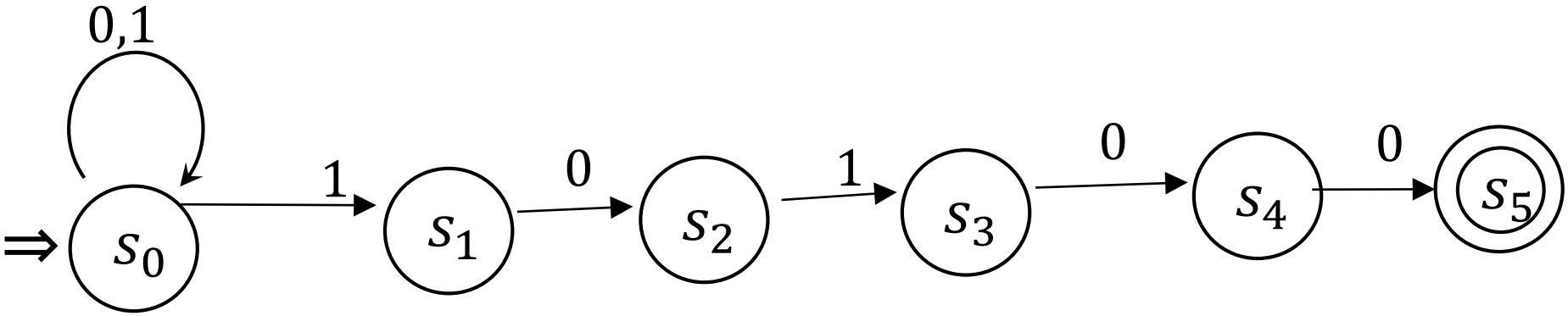
Hint: 上記の言語は全て、2つのより簡単な言語の共通部分である

3.非決定性有限オートマトン (1)

今までのfa (決定性有限オートマトン dfa):
現在の状態と、読む入力記号から、一意的に次の状態が定まる。

非決定性有限オートマトン (nfa):
次の状態として二つ以上の可能性があってもよい。

例:　　（次の行先が無くてもよい）



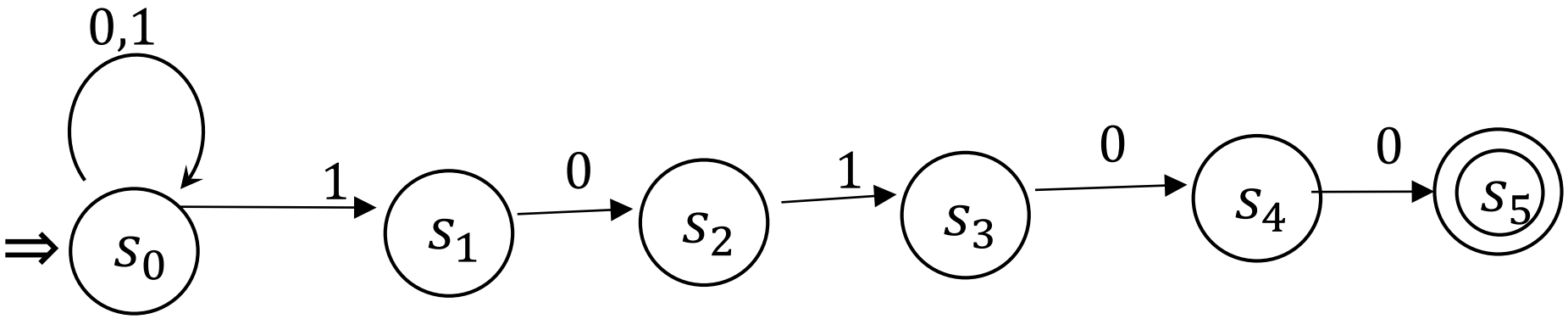
3.非決定性有限オートマトン (2)

定義：非決定性有限オートマトン (nfa)　$M = (K, \Sigma, \delta, s_0, F)$
状態遷移関数δがdfaと異なり、状態の集合への写像になる：

$$\delta: K \times \Sigma \to 2^K$$

2^K：べき集合(Kの全ての部分集合からなる集合）

例



$\delta(s_0, 1) = \{s_0, s_1\}, \delta(s_1, 1) = \emptyset, \delta(s_2, 0) = \emptyset$, …など。

3.非決定性有限オートマトン (3)

非決定性有限オートマトン (nfa)　$M = (K, \Sigma, \delta, s_0, F)$
状態遷移関数δ: $K \times \Sigma \to 2^K$ に対して、

写像δ':　$K \times \Sigma^* \to 2^K$を以下のように再帰的に定義する:

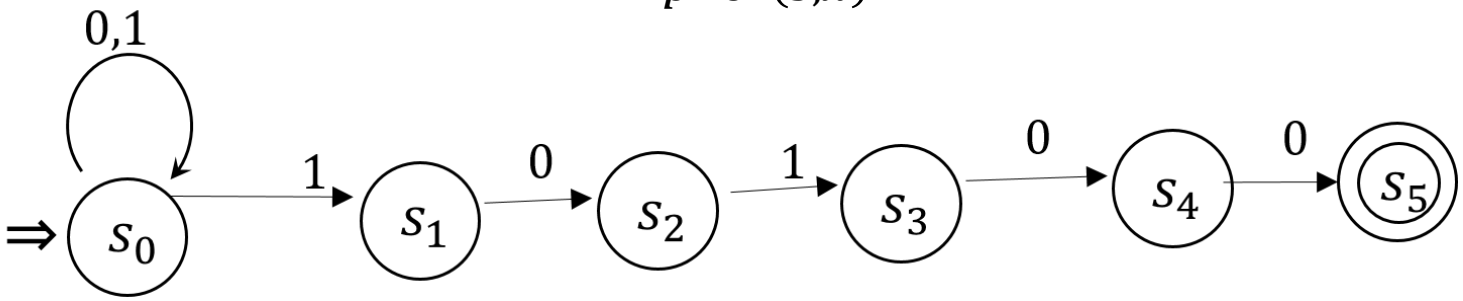
$\forall s \in K, \forall x \in \Sigma^*, \forall a \in \Sigma$に対して、

① $\delta'(s, \epsilon) = \{s\}$

② $\delta'(s, xa) = \{q \in K \mid \exists p \in K \ (p \in \delta'(s, x)$かつ$q \in \delta(p, a))\}$

$$= \bigcup_{p \in \delta'(s,x)} \delta(p, a)$$

例

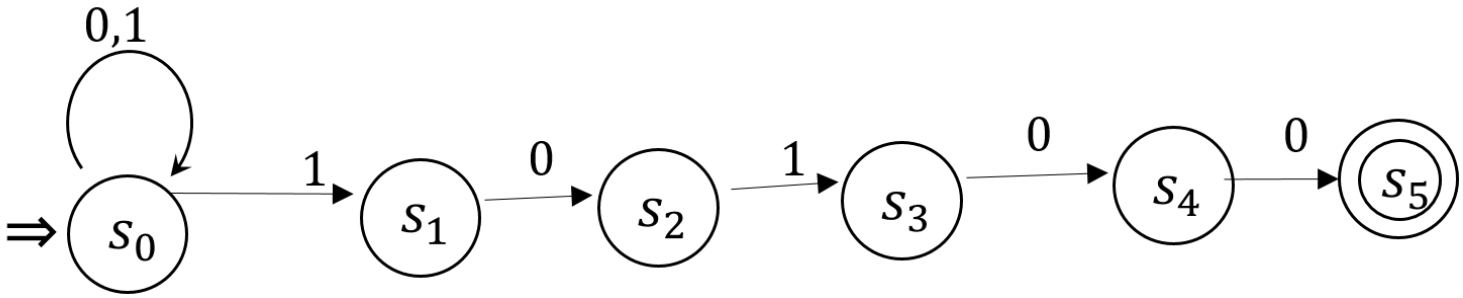


3.非決定性有限オートマトン (4)

① $\delta'(s, \epsilon) = \{s\}$

② $\delta'(s, xa) = \{q \in K \mid \exists p \in K \left(p \in \delta'(s, x) \text{かつ} q \in \delta(p, a)\right)\}$

例



$\delta'(s_0, 101) = \{q \in K \mid \exists p \in K \left(p \in \delta'(s_0, 10) \wedge q \in \delta(p, 1)\right)\}$

$\delta'(s_0, 10) = \{q \in K \mid \exists p \in K \left(p \in \delta'(s_0, 1) \wedge q \in \delta(p, 0)\right)\}$

$= \{q \in K \mid \exists p \in K \left(p \in \{s_0, s_1\} \wedge q \in \delta(p, 0)\right)\}$

$= \delta(s_0, 0) \cup \delta(s_1, 0) = \{s_0, s_2\}$

$\delta'(s_0, 101) = \{q \in K \mid \exists p \in K \left(p \in \{s_0, s_2\} \wedge q \in \delta(p, 1)\right)\}$

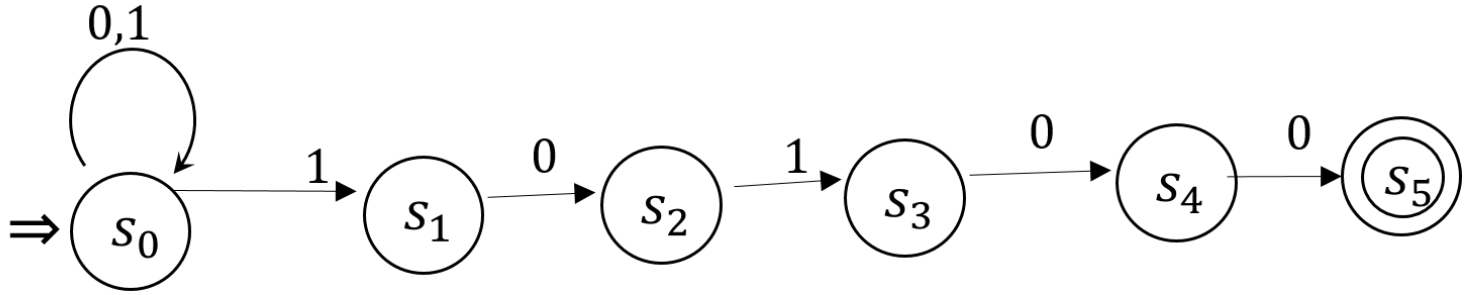
$= \delta(s_0, 1) \cup \delta(s_2, 1) = \{s_0, s_1, s_3\}$

3.非決定性有限オートマトン (5)

$\delta'(s_0, 1) = \{s_0, s_1\}$を
『s_0で1を読むと状態s_0とs_1のいずれにも行ける』と解釈する。

$\delta'(s_0, 101) = \{s_0, s_1, s_3\}$ は、
『s_0で101を読むと状態s_0とs_1とs_3のいずれにも行ける』と解釈する。

$\delta'(s_1, 1) = \emptyset$は
『s_1で1を読むと行ける状態が存在しない』と解釈する。



3.非決定性有限オートマトン (6)

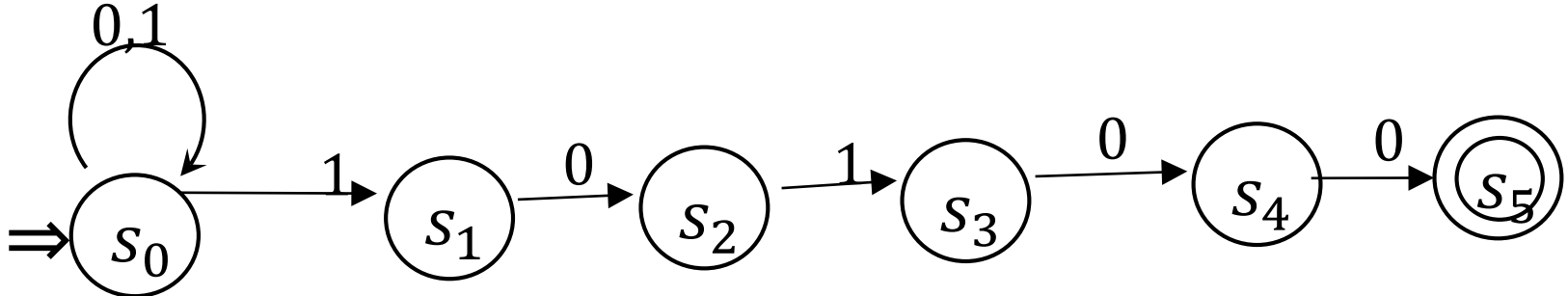
定義：非決定性有限オートマトン (nfa) $M = (K, \Sigma, \delta, s_0, F)$ と列xに対して、$\delta'(s_0, x) \cap F \neq \emptyset$のとき、

つまり、$\delta'(s_0, x)$に少なくとも一つの受理状態が含まれるときに、

『Mはxを受理する』と呼ぶ。

要するに、初期状態から列xを読んである受理状態に到達することができるなら『受理』である。

例： 最後が“10100”で終わる列の全体を受理するオートマトン

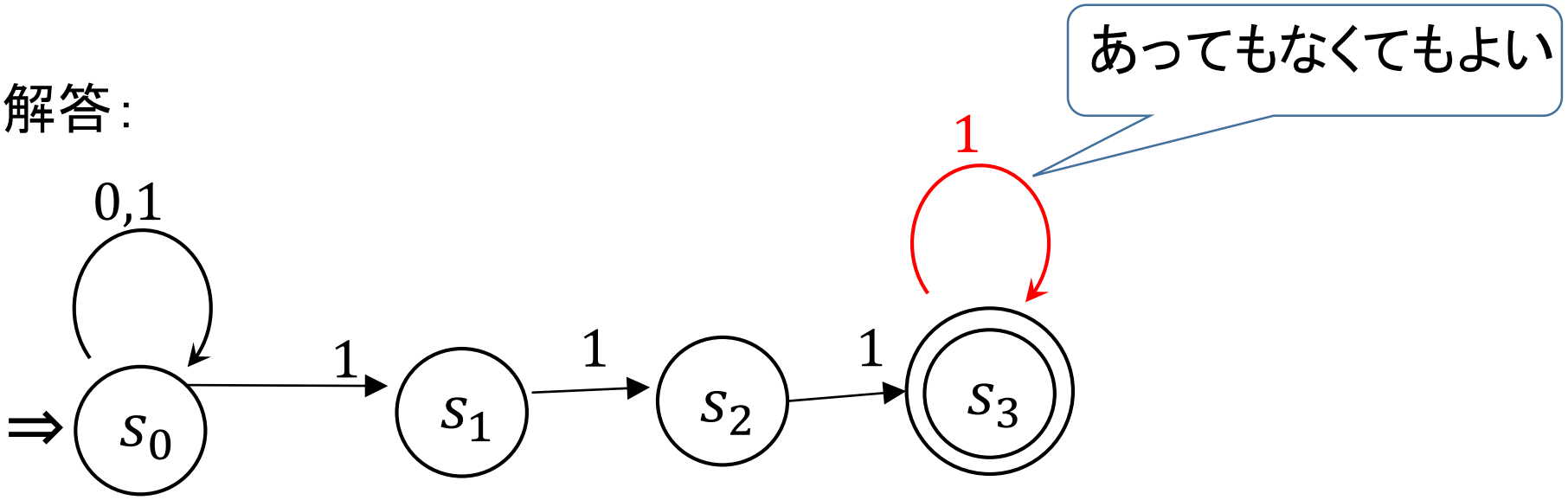


3.非決定性有限オートマトン (7)

例題2.7

言語 $\{x111 \mid x \in \{0,1\}^*\}$ を認識するnfaを与えよ。

解答:



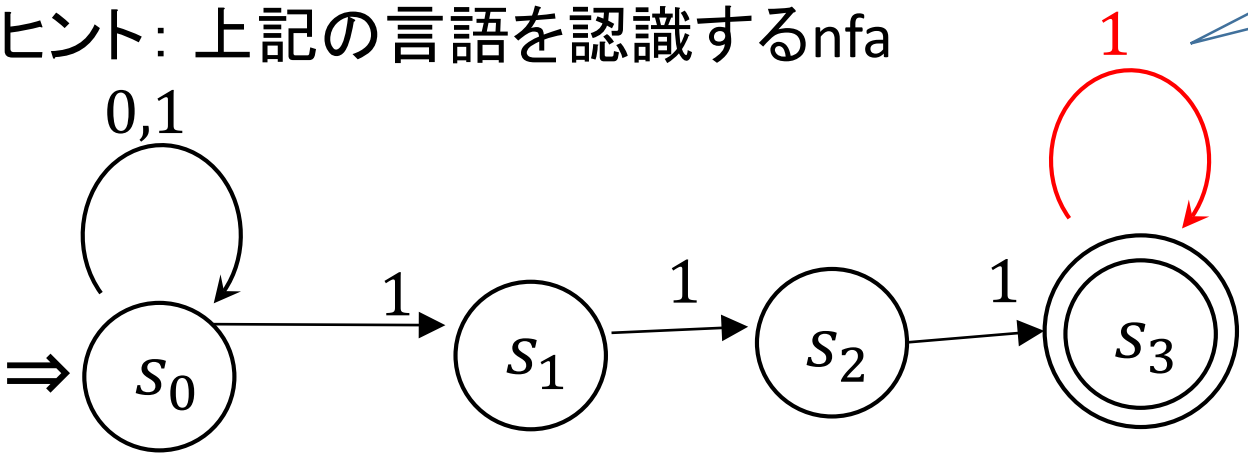
3.非決定性有限オートマトン (8)

練習問題:

言語 $\{x111 \mid x \in \{0,1\}^*\}$ を認識するdfa(決定性有限オートマトン)を与えよ。

あってもなくてもよい

ヒント: 上記の言語を認識するnfa

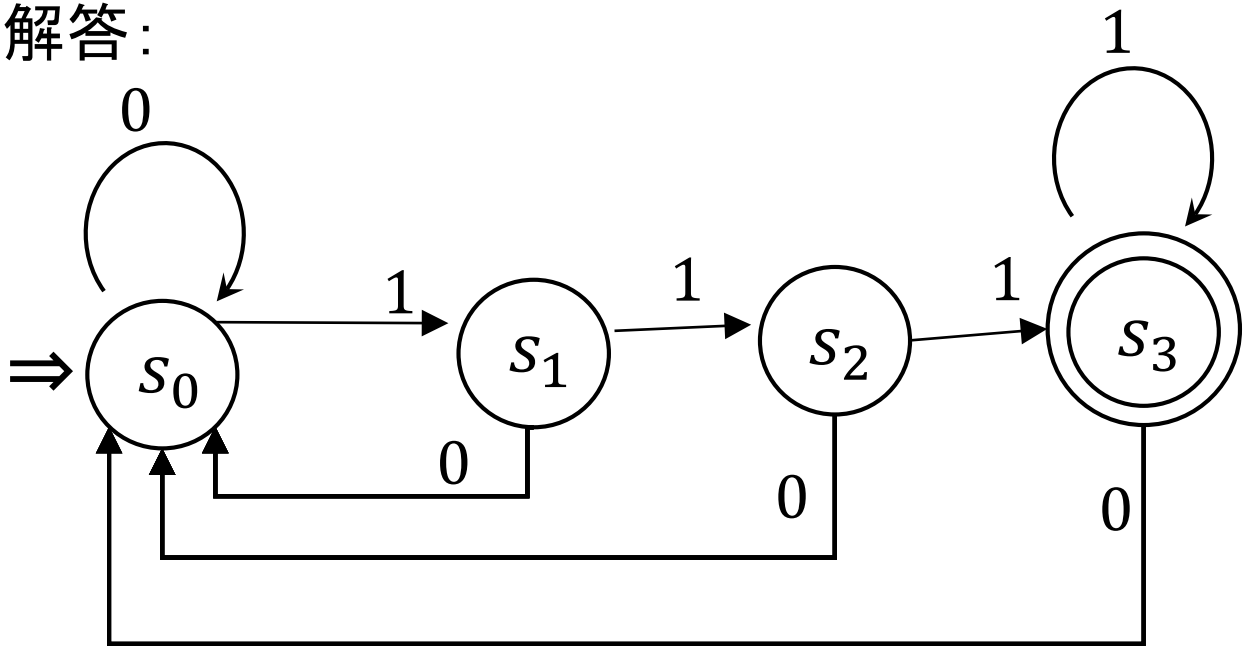


3.非決定性有限オートマトン (9)

練習問題:

言語 $\{x111 \mid x \in \{0,1\}^*\}$ を認識するdfa(決定性有限オートマトン)を与えよ。

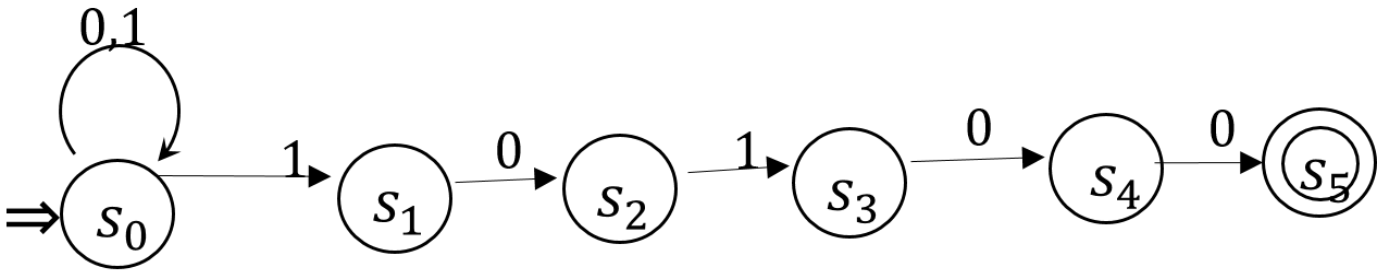
解答:



3.非決定性有限オートマトン(10)

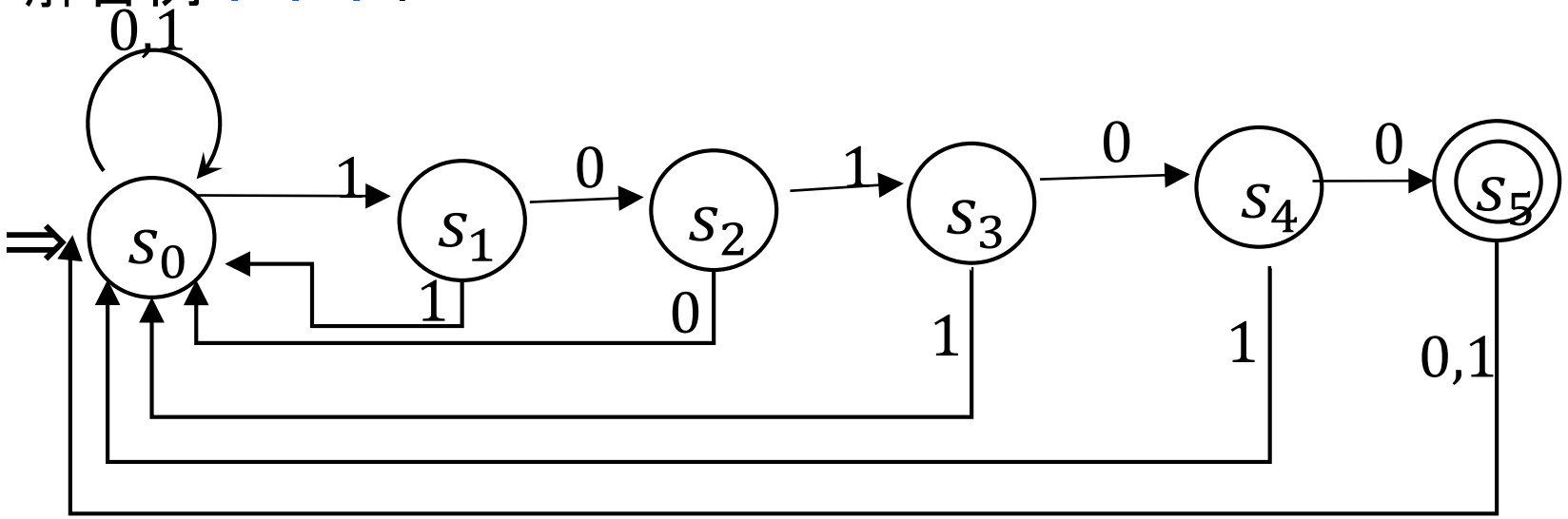
例題:

nfa



が認識する言語(10100で終わる列すべて)を認識するdfa(決定性有限オートマトン)を与えよ。

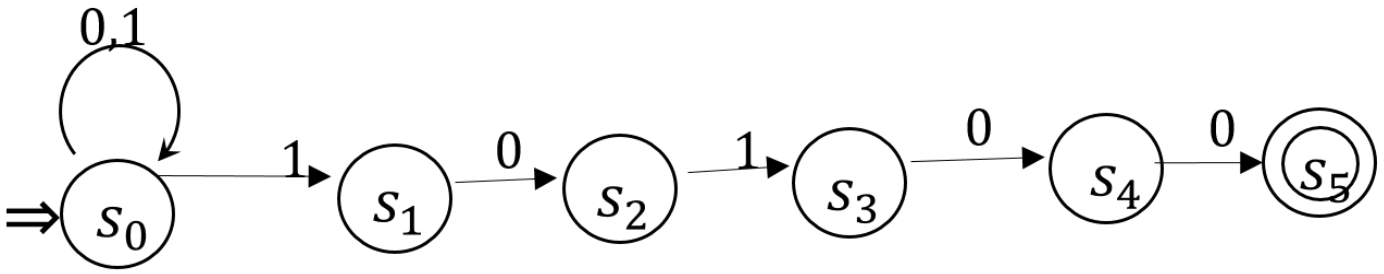
解答例？？？:



3.非決定性有限オートマトン(11)

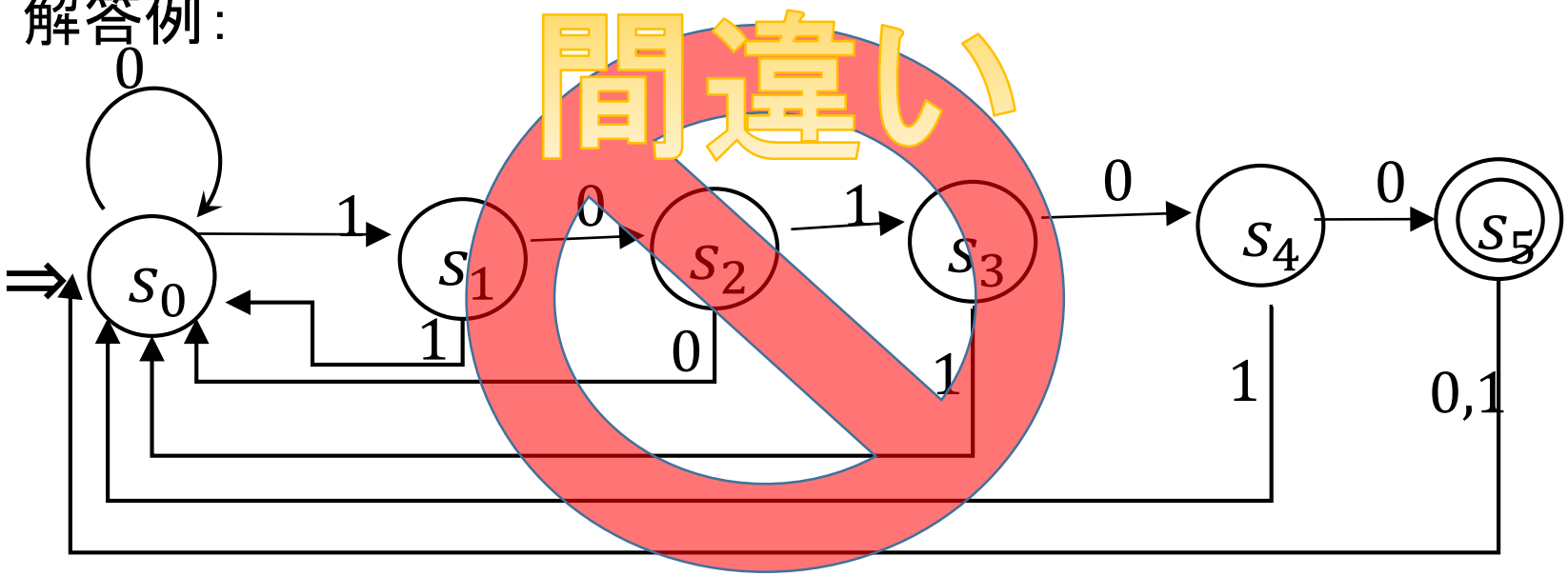
例題:

nfa



が認識する言語(10100で終わる列すべて)を認識するdfa(決定性有限オートマトン)を与えよ。

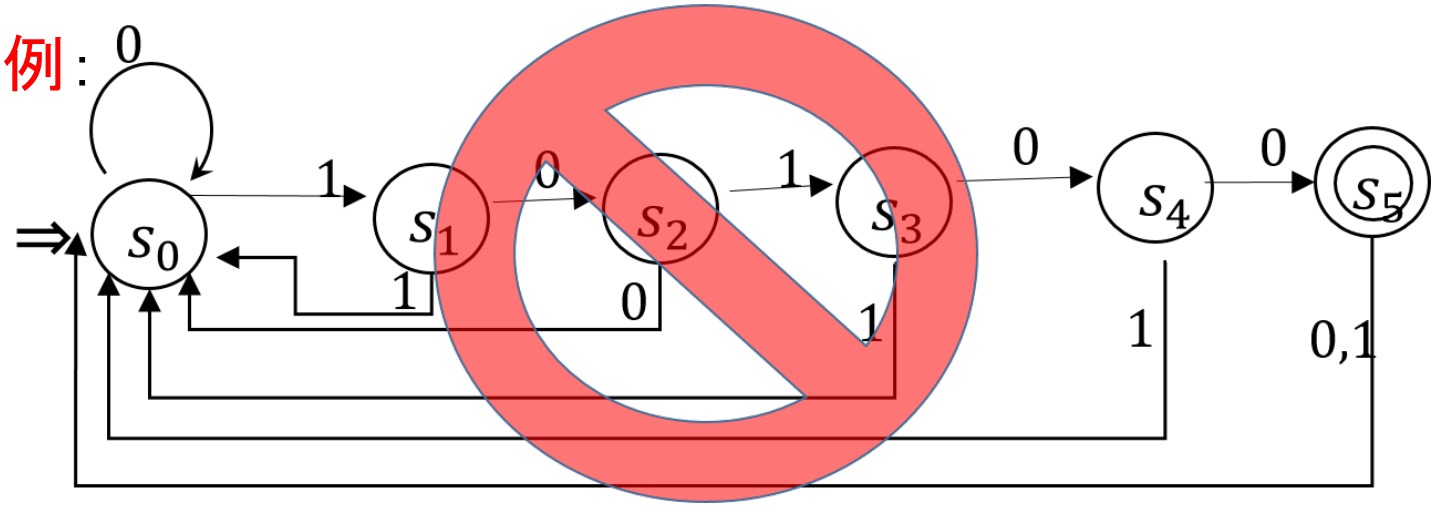
解答例:



3.非決定性有限オートマトン (12)

例題:　10100で終わる列全てからなる言語を認識するdfaを与えよ。

間違った解答例:



例えば、言語に入っている1010100を読むと

$s_0 \to s_1 \to s_2 \to s_3 \to s_4 \to s_0 \to s_0 \to s_0$となり受理されない。

$s_4 \to s_0$がおかしい。

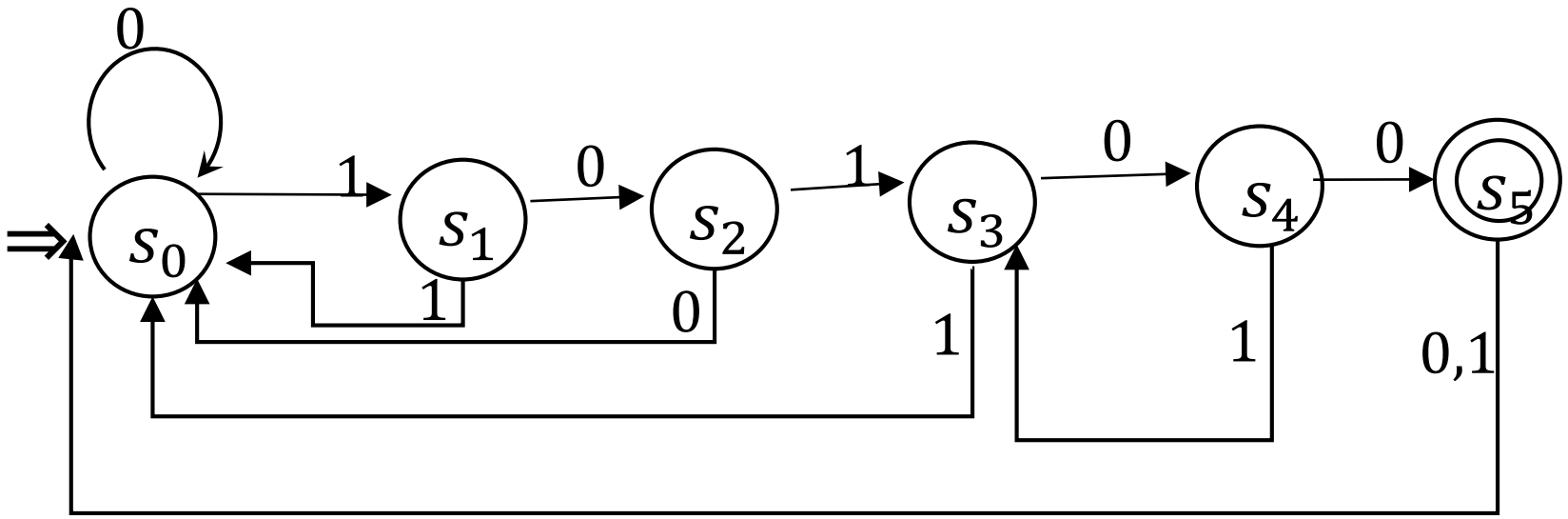
s_1は1を、s_2は10を、s_3は101を、s_4は1010を直前に読んだことを記憶するための状態である。

s_4で1を読むと、10101を読んだことになるので、s_0ではなくs_3に行かなくてはならない。

3.非決定性有限オートマトン (13)

例題:　10100で終わる列全てからなる言語を認識するdfaを与えよ。

改良した解答例？？:



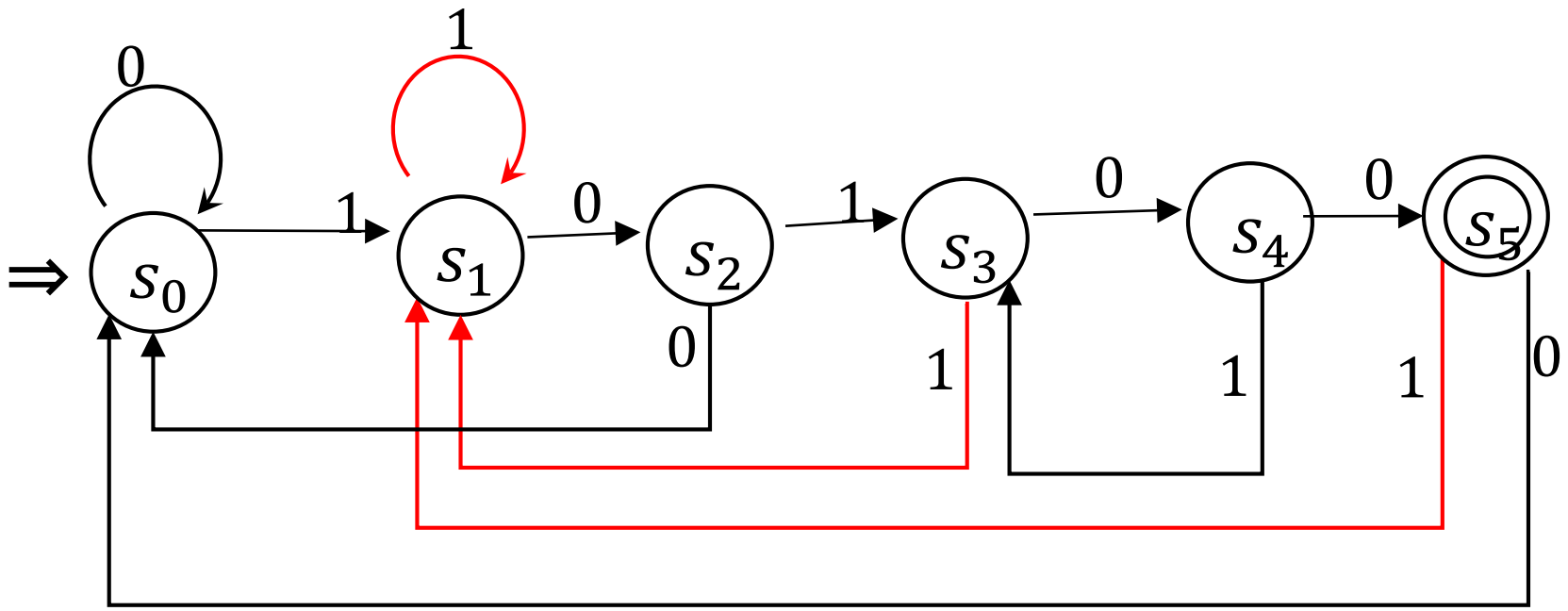
まだ、三つ間違いが残っている。

s_1は1を、s_2は10を、s_3は101を、s_4は1010を直前に読んだことを記憶するための状態である。

3.非決定性有限オートマトン (15)

例題:　10100で終わる列全てからなる言語を認識するdfaを与えよ。

本当に正しい解答例



s_1は1を、s_2は10を、s_3は101を、s_4は1010を直前に読んだことを記憶するための状態である。

3.非決定性有限オートマトン (16)

今までのfaの1ステップは、以下の三つの動作からなる:

① 入力を読む

② ヘッドを(1コマ右に)動かす

③ 状態を変える(状態遷移関数にしたがって)

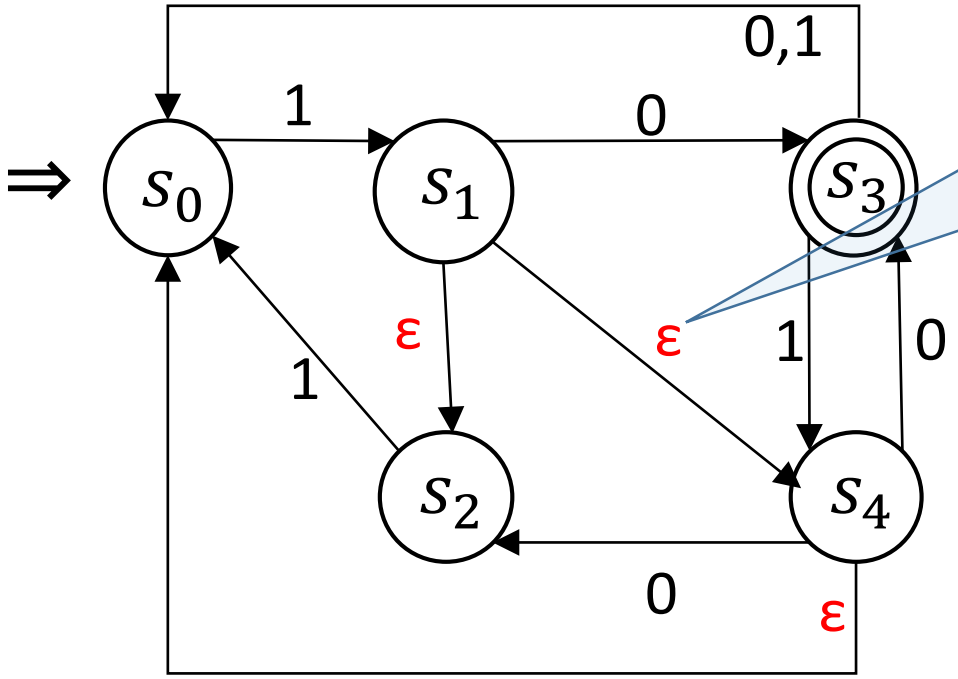
『②ヘッドを動かす』ことをしないことを許す、

つまり『入力を読まずに状態を変えることができる』オートマトンを考える:

ε入力付非決定性有限オートマトン (εnfa)

3.非決定性有限オートマトン (17)

ε入力付非決定性有限オートマトン (εnfa)



s_1から入力を読まずに
（ヘッドを動かさずに）
s_4へ 遷移することができる。

状態がs_1でヘッドの下の記号が0なら、s_3に行ってヘッドを右に一つ動かす他に、ヘッドを動かさずにs_2, s_4に遷移することも可能。

3.非決定性有限オートマトン (18)

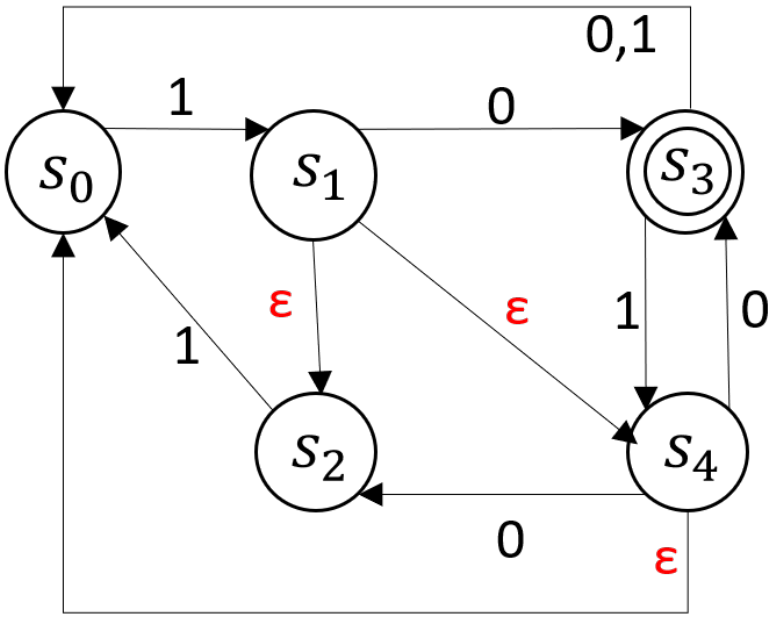
定義ε入力付非決定性有限オートマトン (εnfa)

$M = (K, \Sigma, \delta, s_0, F)$

nfaとの違いは、δの定義域が、

$K \times (\Sigma \cup \{\epsilon\})$となっていること。

⇒



つまり、$\delta: K \times (\Sigma \cup \{\epsilon\}) \to 2^K$

右の例なら、

$\delta(s_1, 0) = \{s_3\}$, $\delta(s_1, \epsilon) = \{s_2, s_4\}$,...

テープを読み終わったときに、受理状態に到達できれば受理される。

例では、1110も$s_0 \to s_1 \to s_2 \to s_0 \to s_1 \to s_3$とすれば受理される。

3.非決定性有限オートマトン (18)

オートマトンの動作途中の状況を表すために、$x \in \Sigma^*, p \in K$に対して様相 (x, p) を定義する。

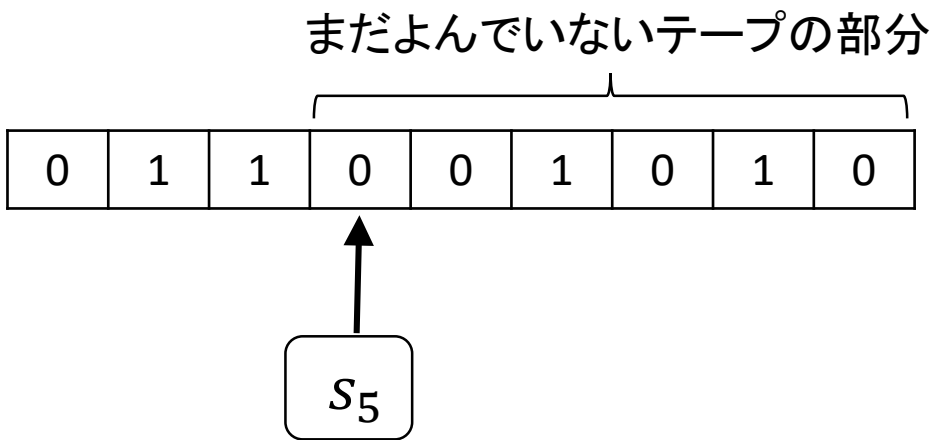
xはテープのまだ読んでいない部分の列

pはそのときの状態

読み終わった部分は将来の動作に影響しない。

例:

様相$(001010, s_5)$



3.非決定性有限オートマトン (19)

定義： 以下の(i) もしくは (ii) を満たすとき、様相$c_1 = (x_1, p_1)$から様相$c_2 = (x_2, p_2)$へ1ステップで移れる （$c_1 \Rightarrow c_2$と書く)と呼ぶ：

(i) $x_1 = x_2$かつ$p_2 \in \delta(p_1, \epsilon)$

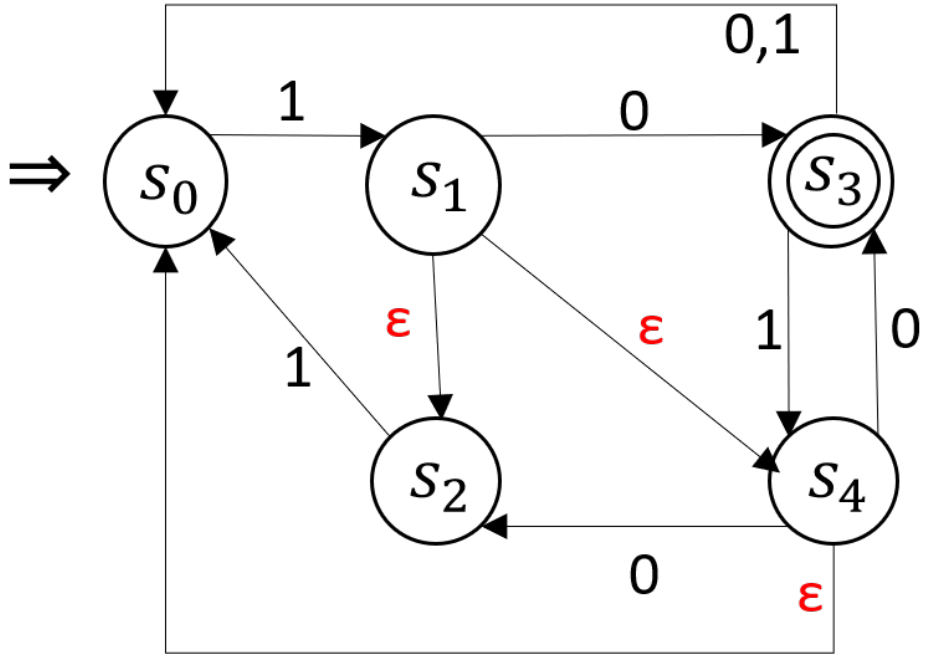
(ii) $\exists a \in \Sigma$ に対して、$x_1 = ax_2$ かつ $p_2 \in \delta(p_1, a)$

例：

$(00011, s_1) \Rightarrow (0011, s_3)$

$(00011, s_1) \Rightarrow (00011, s_4)$

である。



3.非決定性有限オートマトン (20)

定義：$c_0 \Rightarrow c_1 \Rightarrow \cdots \Rightarrow c_k$のとき、
様相c_0からc_kへ何ステップかで移れる ($c_0 \overset{*}{\Rightarrow} c_k$と書く)と呼ぶ。
0ステップの場合も含む（つまり$c_0 = c_k$のとき。）

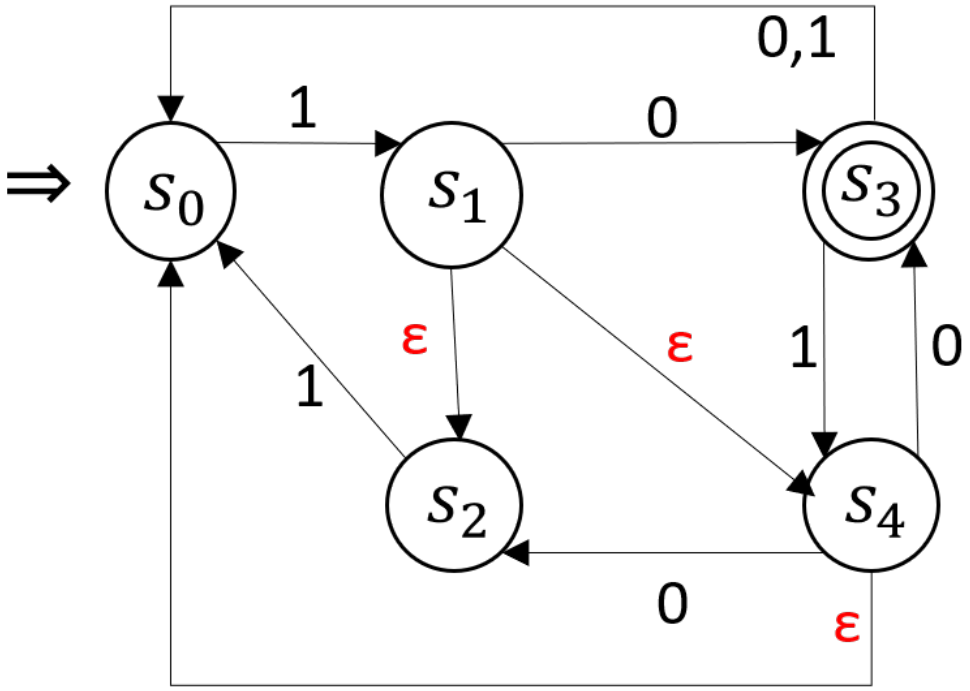
定義：列$x \in \Sigma^*$は、初期状態s_0、ある受理状態s_Fに対し、
$(x, s_0) \overset{*}{\Rightarrow} (\epsilon, s_F)$であるとき受理されると呼ぶ。

この定義より、最後がϵ入力でも受理される：
$(x, s_0) \overset{*}{\Rightarrow} (\epsilon, s) \Longrightarrow (\epsilon, s_F)$ で、$s \notin F$でもxは受理される。

3.非決定性有限オートマトン (21)

練習問題:

左図のオートマトンで、列110が受理されることを示せ。

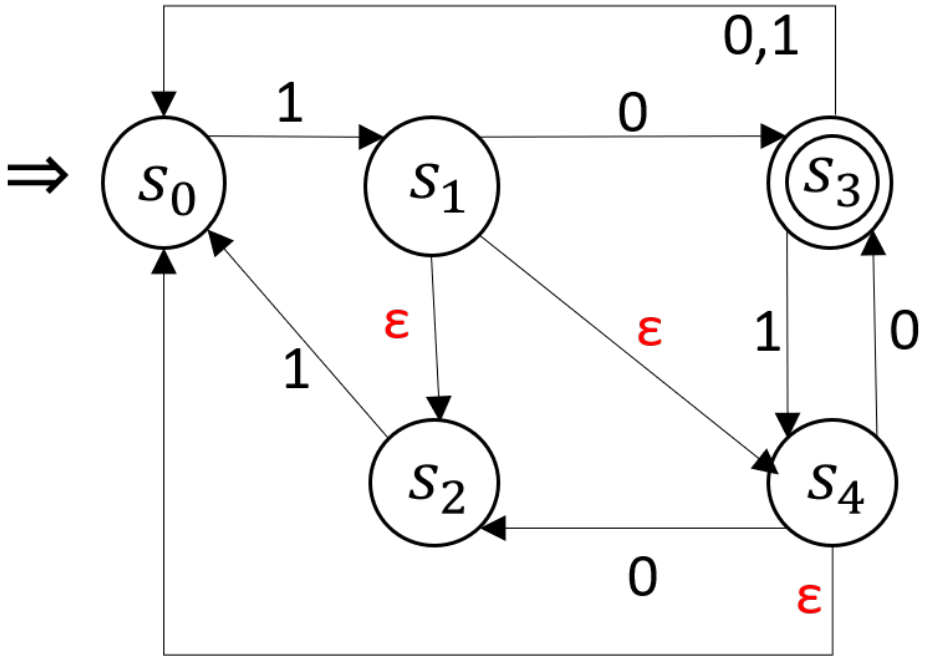


ヒント: $(110, s_0) \Rightarrow \cdots \Rightarrow (\epsilon, s_3)$の形の様相の列を作ればよい。

3.非決定性有限オートマトン (22)

練習問題:

左図のオートマトンで、列110が受理されることを示せ。



(解答例)

$(110, s_0) \Rightarrow (10, s_1) \Rightarrow (10, s_4) \Rightarrow (10, s_0) \Rightarrow (0, s_1) \Rightarrow (\epsilon, s_3)$

4. 有限オートマトンと正規表現の等価性 (1)

定義　正規表現を持つ言語を正規言語と呼ぶ。

この節の目的:

(決定性)有限オートマトン(dfa)と正規表現が等価であることを示す。

⇔任意の言語Lに対して、Lがdfaで認識可能であることは、
Lが正規言語であることの必要十分条件

証明の方針 : dfa、nfa、εnfa、正規言語の集合を、

$\mathcal{L}(\mathrm{dfa})$、$\mathcal{L}(\mathrm{nfa})$、$\mathcal{L}(\epsilon n\mathrm{fa})$、$\mathcal{L}$(正規言語)と書く。

$$\mathcal{L}(\mathrm{dfa}) = \mathcal{L}(\mathrm{nfa}) = \mathcal{L}(\epsilon n\mathrm{fa}) = \mathcal{L}(\text{正規言語})$$

を証明することを目指す。

４. 有限オートマトンと正規表現の等価性 (2)

$\mathcal{L}(\mathrm{dfa}) = \mathcal{L}(\mathrm{nfa}) = \mathcal{L}(\epsilon n\mathrm{fa}) = \mathcal{L}(\text{正規言語})$

を証明することを目指す。

dfaは特殊なnfa、nfaは特殊なεnfaなので、

$\mathcal{L}(\mathrm{dfa}) \subseteq \mathcal{L}(\mathrm{nfa}) \subseteq \mathcal{L}(\epsilon n\mathrm{fa})$は明らか。

『nfaがdfaで模倣できる』こと($L \in \mathcal{L}(\mathrm{nfa}) \Rightarrow L \in \mathcal{L}(\mathrm{dfa})$)と、
(⇔nfaの動作をfaで「真似」して同じ言語を認識するようにできる)
『εnfaがnfaで模倣できる』こと($L \in \mathcal{L}(\epsilon\mathrm{nfa}) \Rightarrow L \in \mathcal{L}(\mathrm{nfa})$)
を証明した後で、

$\mathcal{L}(\epsilon n\mathrm{fa}) = \mathcal{L}(\text{正規表現})$を証明すればよい。

4. 有限オートマトンと正規表現の等価性 (3)

補題2.1 言語Lがnfaで認識できるならば、dfaでも認識できる。

(証明の概要)
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に直せることを示す。

<アイデア：部分集合構成>

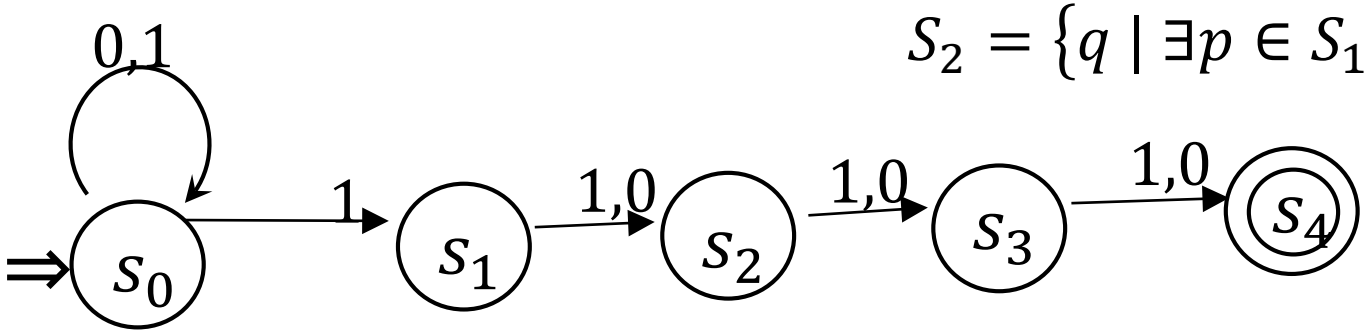
Nにテープを途中まで読ませたときに、到達可能な状態の集合を$S_1 \subseteq K$とする。ここから、さらに記号aを読んだときに到達できる状態の集合S_2は以下で与えられる。

$$S_2 = \{q \mid \exists p \in S_1 (q \in \delta(p, a))\}$$

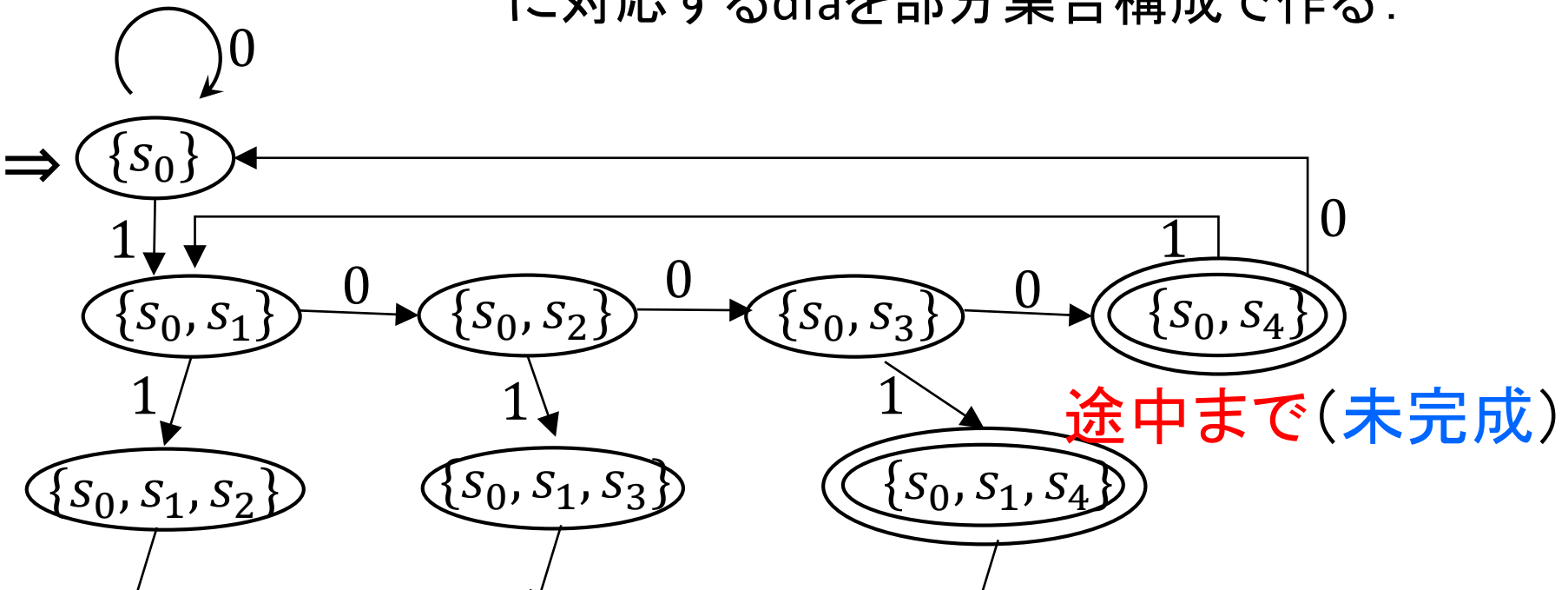
S_1とS_2を単独の状態とみなす(群状態)ことで、S_1からS_2へ入力記号aを読むことで決定的に遷移するdfa Dが得られる。
初期状態は$\{s_0\}$とし、Nの受理状態を一つでも含む状態集合をDの受理状態とする。

4. 有限オートマトンと正規表現の等価性 (4)

$$S_2 = \{q \mid \exists p \in S_1 (q \in \delta(p, a))\}$$



に対応するdfaを部分集合構成で作る:



途中まで(未完成)

4. 有限オートマトンと正規表現の等価性 (6)

補題2.2 言語Lがεnfaで認識できるならば、nfaでも認識できる。

(証明の概要) 与えられたεnfa Eに対し、それが認識する言語を変えずにnfa Nに直せることを示す。

<アイデア>

状態は同じで、遷移"$\xrightarrow{\epsilon}$"を消して、通常の遷移"$\xrightarrow{a}$"で置き換えたい

Eの状態sに対して、sから入力信号を読まないで($\xrightarrow{\epsilon}$で)遷移できる状態の集合を$S(s)$とする($s \in S(s)$とする)。 $(s, S(s))$をsに対応するNの状態とする。 ある$t_1 \in S(s)$に対して、t_1からt_2へ記号$a(\neq \epsilon)$によるEの遷移が存在するときに限って、状態$(s, S(s))$から$(t_2, S(t_2))$へ記号aで遷移できるとする。

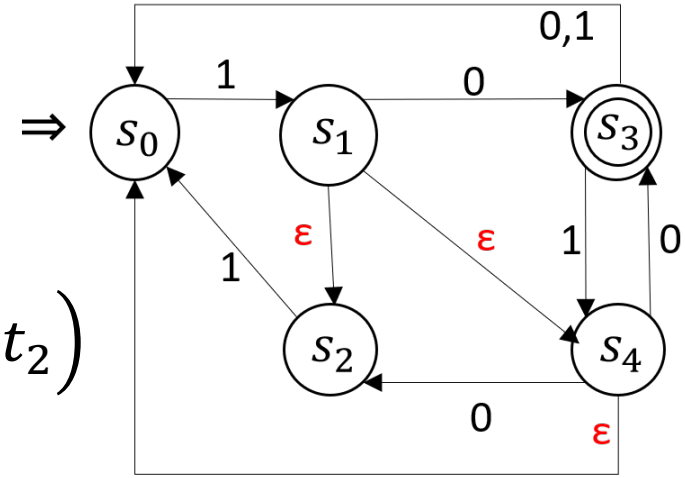
$$(s, S(s)) \xrightarrow{a} (t_2, S(t_2)) \Leftrightarrow \exists t_1 \in S(s)\left(t_1 \xrightarrow{a} t_2\right)$$

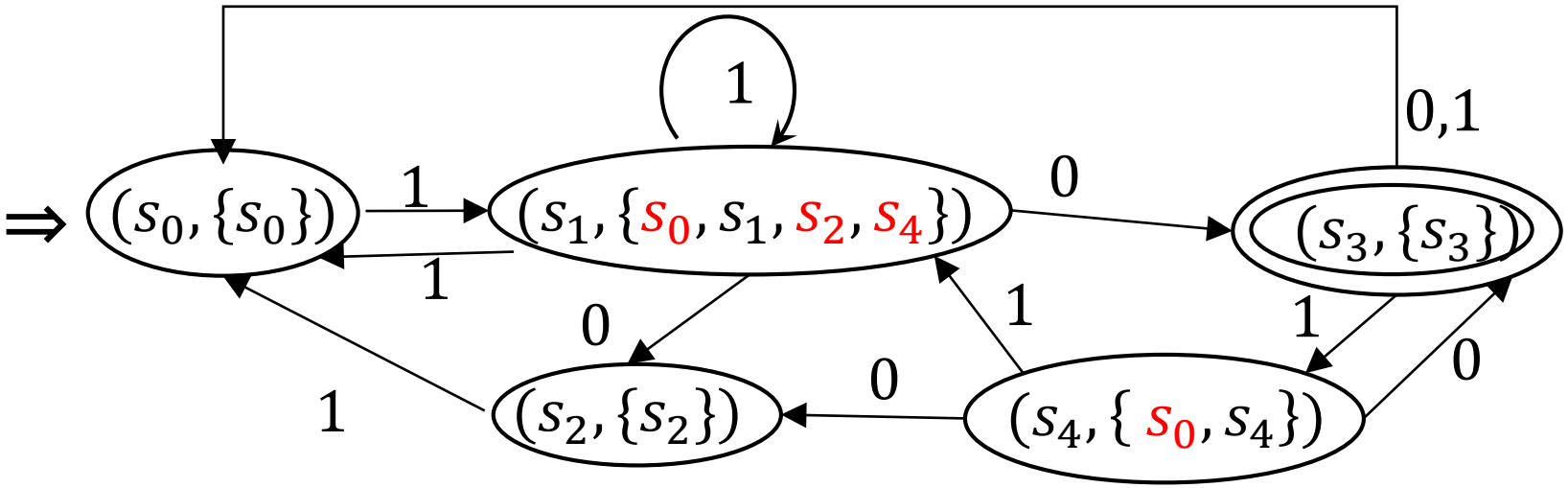
4. 有限オートマトンと正規表現の等価性 (7)

初期状態は$(s_0, S(s_0))$とする。

例 :

$$(s, S(s)) \xrightarrow{a} (t_2, S(t_2)) \Leftrightarrow \exists t_1 \in S(s)\left(t_1 \xrightarrow{a} t_2\right)$$





4. 有限オートマトンと正規表現の等価性 (8)

ここまでで、$L(\mathrm{dfa}) = L(\mathrm{nfa}) = L(\epsilon\mathrm{nfa})$が証明できたので、
$L(正規言語) \subseteq L(\epsilon\mathrm{nfa})$と$L(\mathrm{nfa}) \subseteq L(正規言語)$を示せば完成。

補題 2.3 言語Lが正規言語なら、Lはあるϵnfa Eによって認識される。

証明：一般に、正規表現Rは、別の正規表現R_1とR_2を使って、

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

のどれかの形に書くことができる。

例えば、

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

は、$R_1 = \left(10^* + \left((0 + 111)^* + 10\right)^* + 11^*\right)^*$,
$R_2 = (000 + 111)^* 100^*$に対して$R = R_1 R_2$と書ける。

4. 有限オートマトンと正規表現の等価性 (9)

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

は、$R_1 = \left(10^* + \left((0 + 111)^* + 10\right)^* + 11^*\right)^*$,
$R_2 = (000 + 111)^* 100^*$に対して$R = R_1 R_2$と書ける。
$R_3 = 10^* + \left((0 + 111)^* + 10\right)^* + 11^*$とすると、$R_1 = (R_3)^*$と書ける。

これを続けてどんどんR_4, R_5, R_6 …を作っていくと、
R_5, R_6 …はどんどん簡単になる。

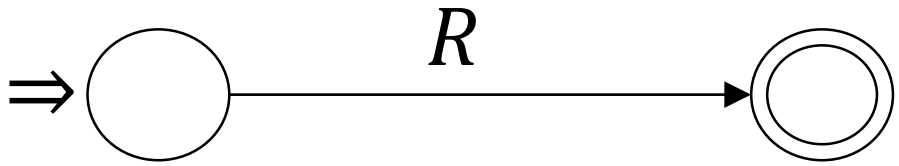
最終的に、これ以上分解できない以下の3通りの正規表現になる:

- 0や1のような1個の記号
- 空列ϵ
- 空集合$\emptyset$

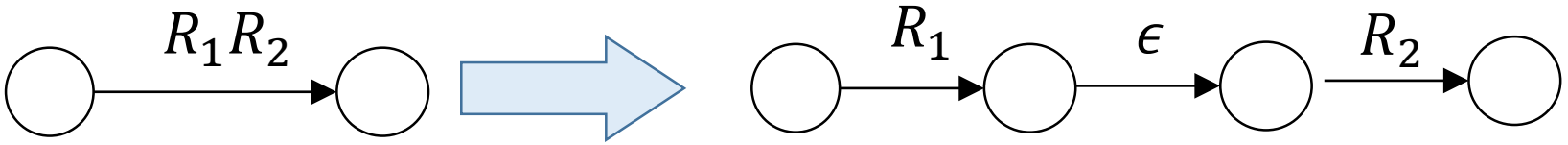
4. 有限オートマトンと正規表現の等価性 (10)

こうして得られた分解を元にRに対応するεnfaを構成する。

1個の初期状態と受理状態を矢で結ぶ

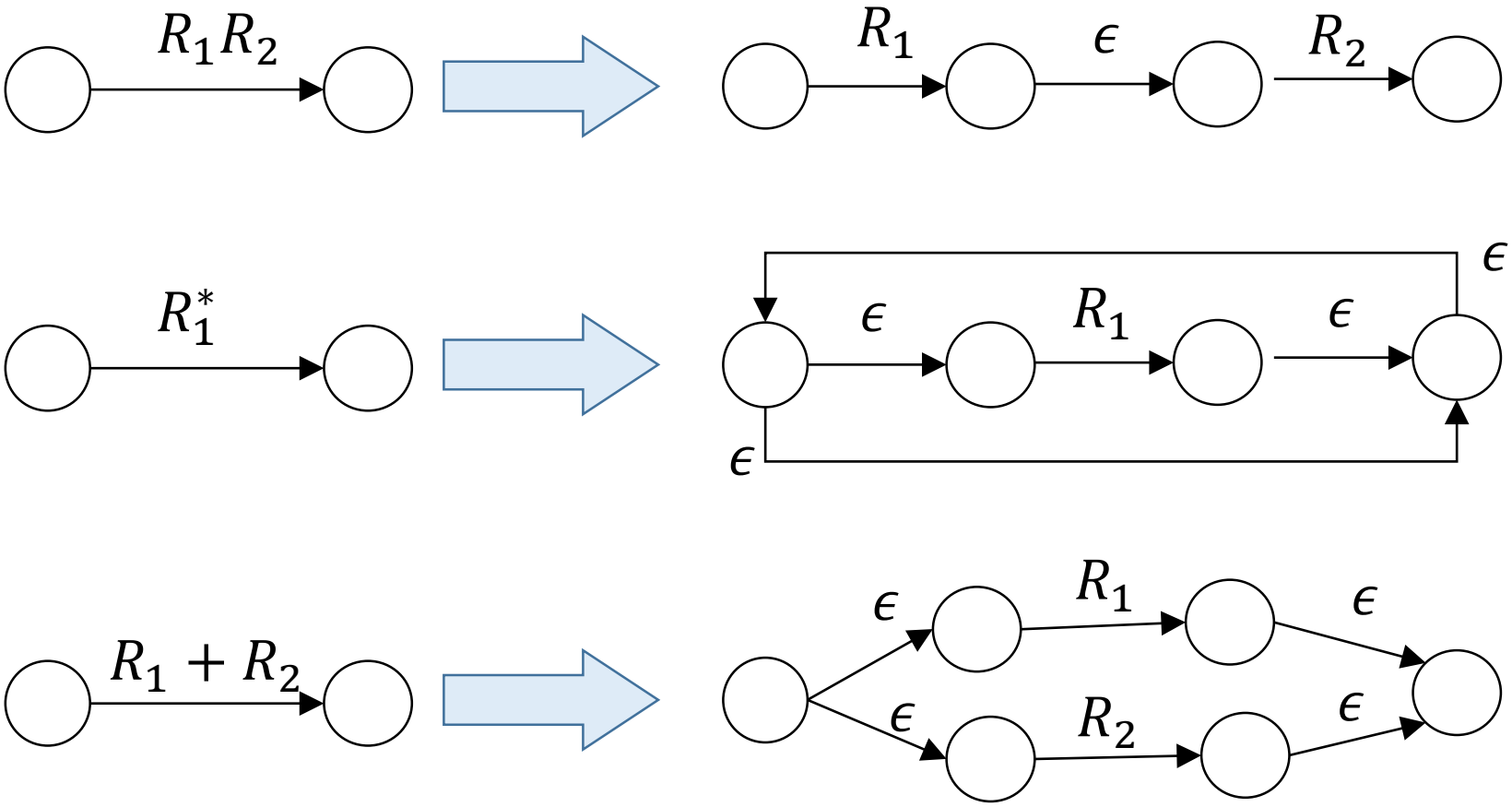


これを、正規表現の分解$R_1 + R_2$，R_1R_2，R_1^*に対応して下記で置き換えていく。



4. 有限オートマトンと正規表現の等価性 (11)

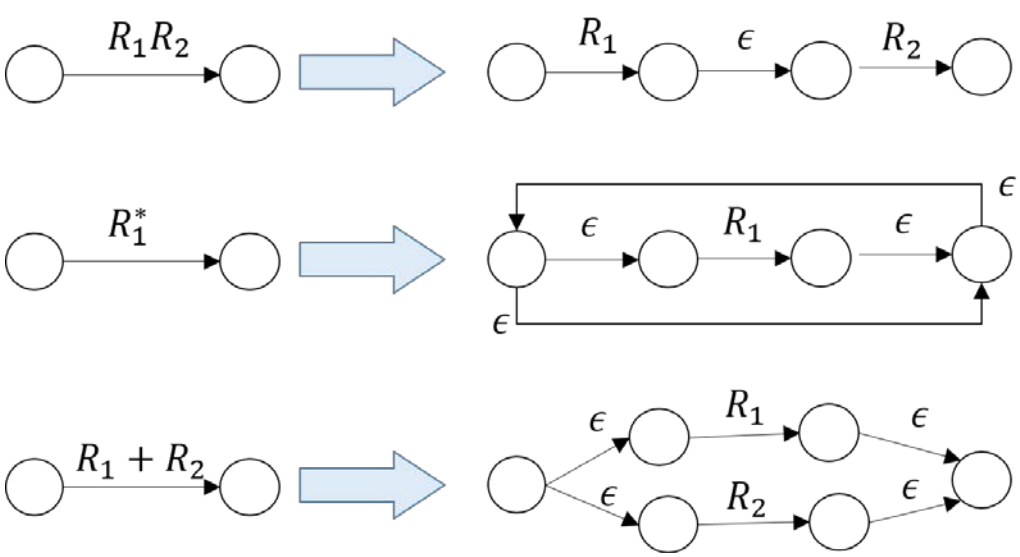
これを、正規表現の分解$R_1 + R_2$，R_1R_2，R_1^*に対応して下記で置き換えていく。



4. 有限オートマトンと正規表現の等価性 (12)

この置き換えを続けていくと、

最終的に各矢の上についているのは、1個の記号、ϵ、$\emptyset$の何れかになる。

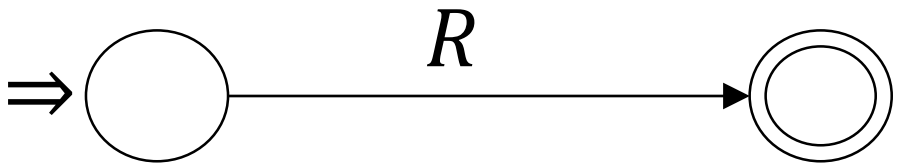


$\emptyset$の場合のみ、その矢を取り去る。
これで、εnfaが完成する。

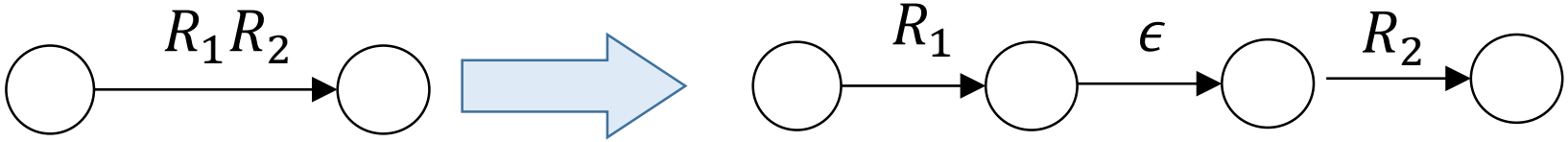
どうしてこの方法が正しいのか？

4. 有限オートマトンと正規表現の等価性 (13)

どうしてこの方法が正しいのか？



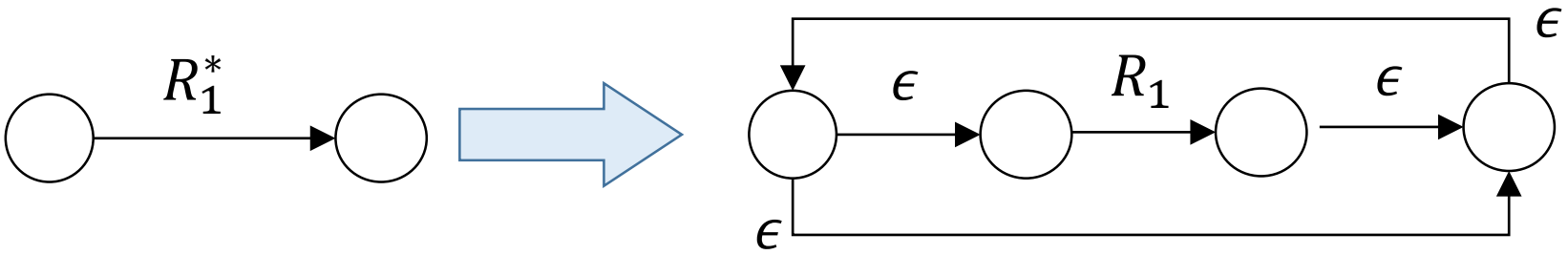
最初の遷移図を、$L(R)$に入る文字列を読むと初期状態から受理状態に遷移できると解釈する。



左は、$L(R_1R_2)$に入る文字列xyを読むと遷移できる

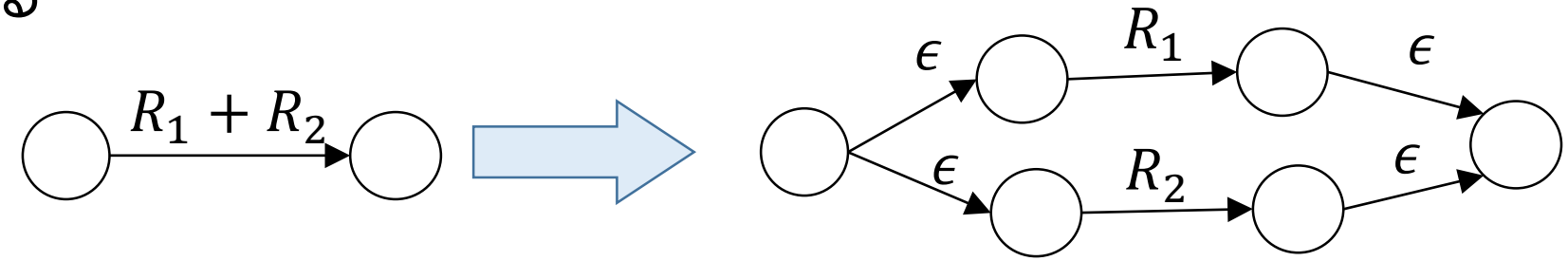
⇒右は、$x \in L(R_1)$を読んで(最初の矢)から、$y \in L(R_2)$を読む(最後の矢)と遷移できる

4. 有限オートマトンと正規表現の等価性 (14)



左は、$L(R_1^*)$に入る文字列を読むと遷移できる

⇒右は、$L(R_1)$に入る文字列を何回か(0回でもよい)読むと遷移できる



左は、$L(R_1 + R_2) = L(R_1) \cup L(R_2)$に入る文字列を読むと遷移

⇒右は、$L(R_1)$に入る文字列もしくは、$L(R_2)$に入る文字列を読むと遷移

４. 有限オートマトンと正規表現の等価性 (１５)

結局、これらの置き換えを行っても、
『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』

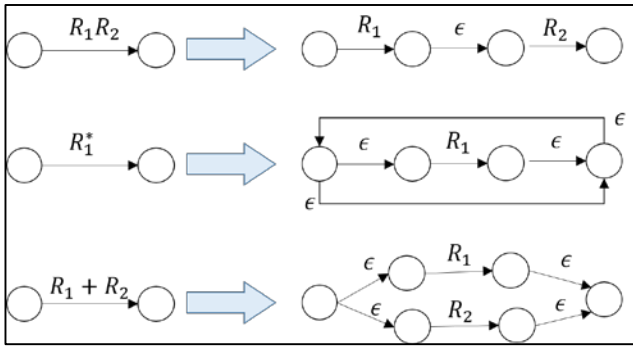
という事実が保存される。

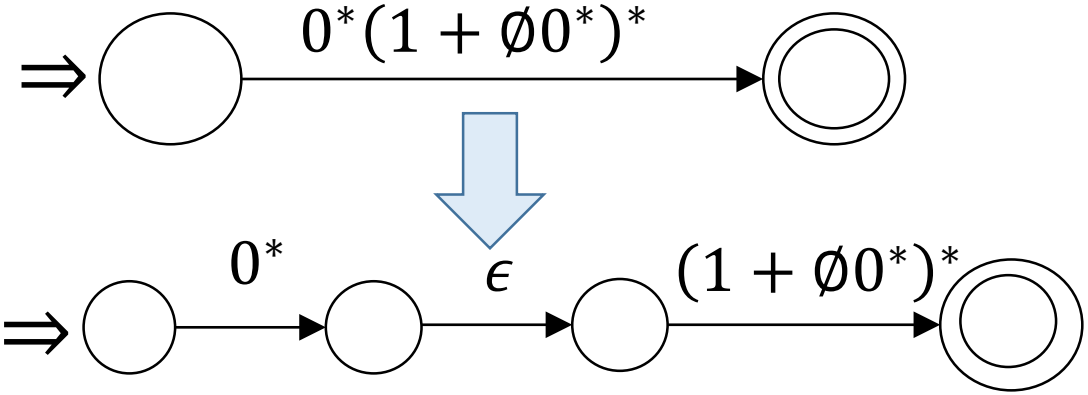
つまり、受理される列は変わらない。

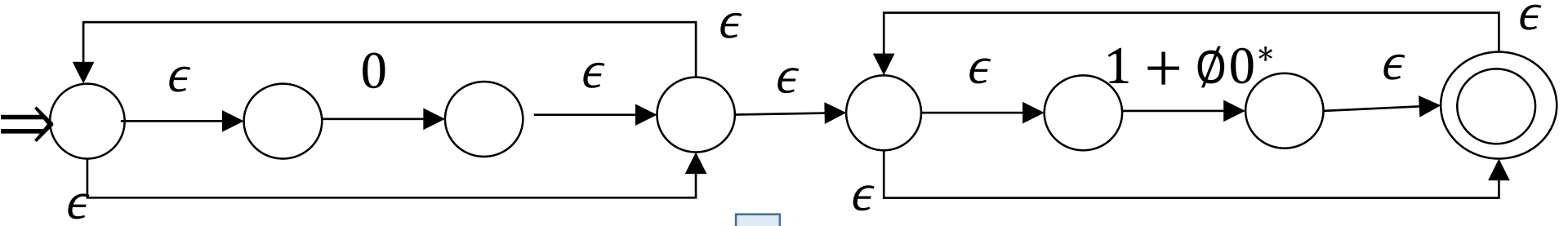
よって、この置き換えを続けて最終的に得られたϵnfaは、言語$L(R)$を認識する。

4. 有限オートマトンと正規表現の等価性 (16)

例：正規表現$0^*(1+\emptyset 0^*)^*$が表現する言語を認識するϵnfa

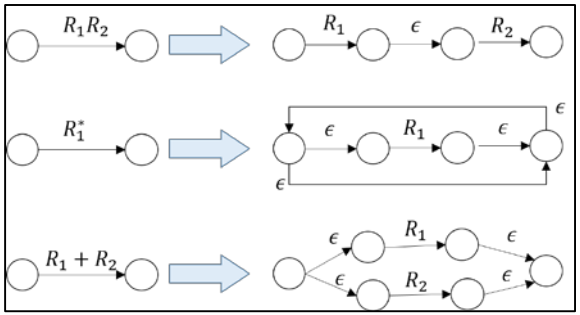


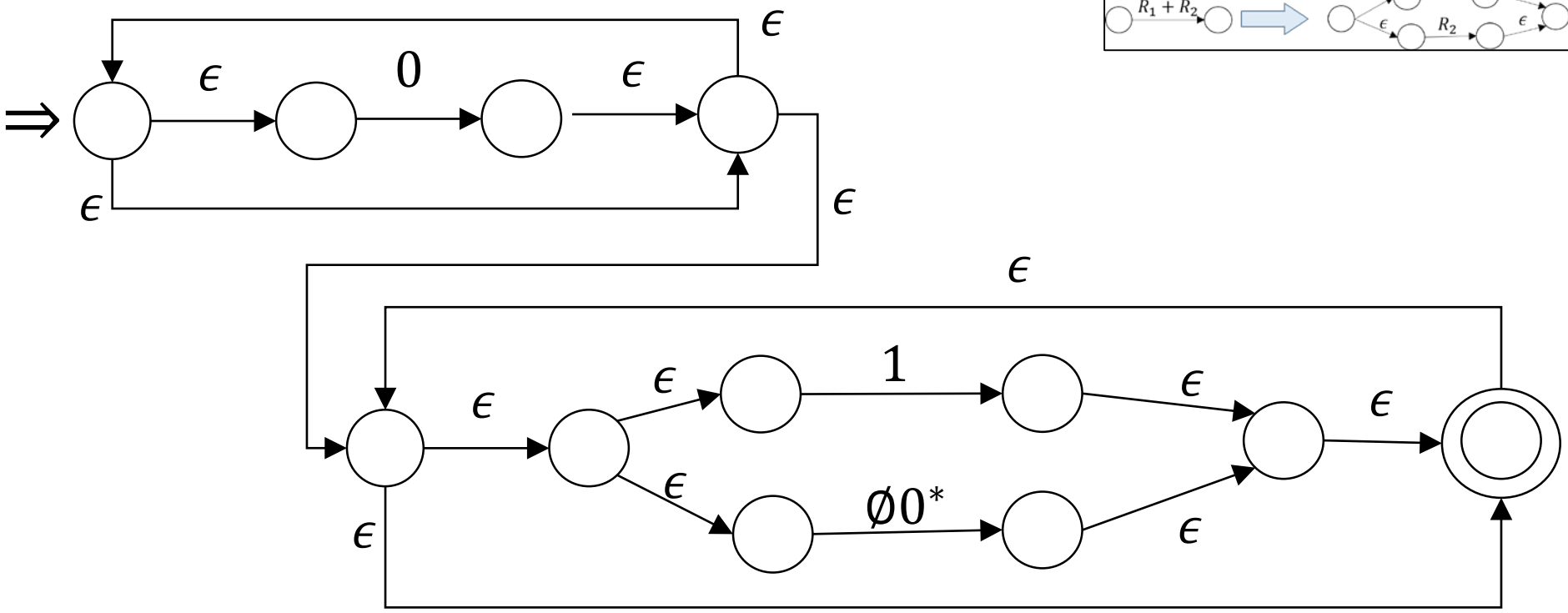




4. 有限オートマトンと正規表現の等価性 (17)

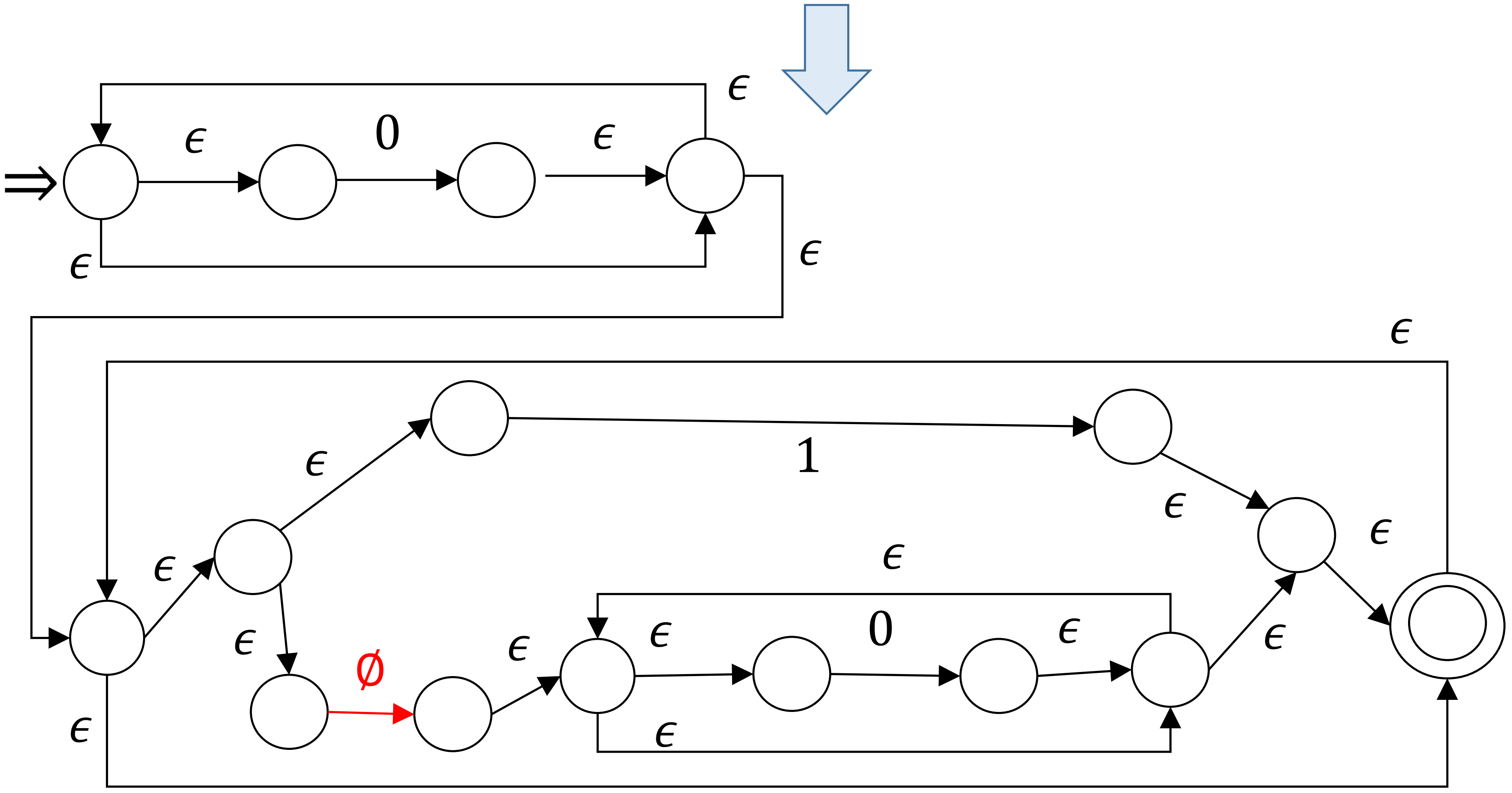
例：正規表現$0^*(1 + \emptyset 0^*)^*$が表現する言語を認識するϵnfa





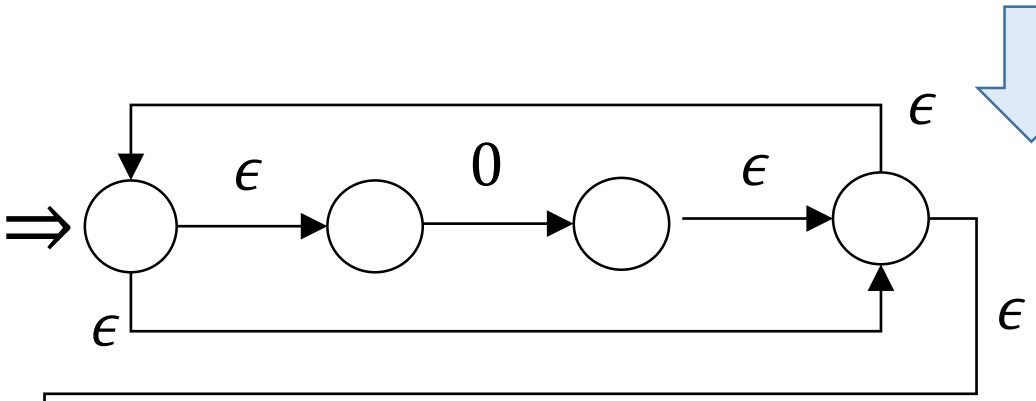
4. 有限オートマトンと正規表現の等価性 (18)

例: 正規表現$0^*(1+\emptyset 0^*)^*$が表現する言語を認識するϵnfa

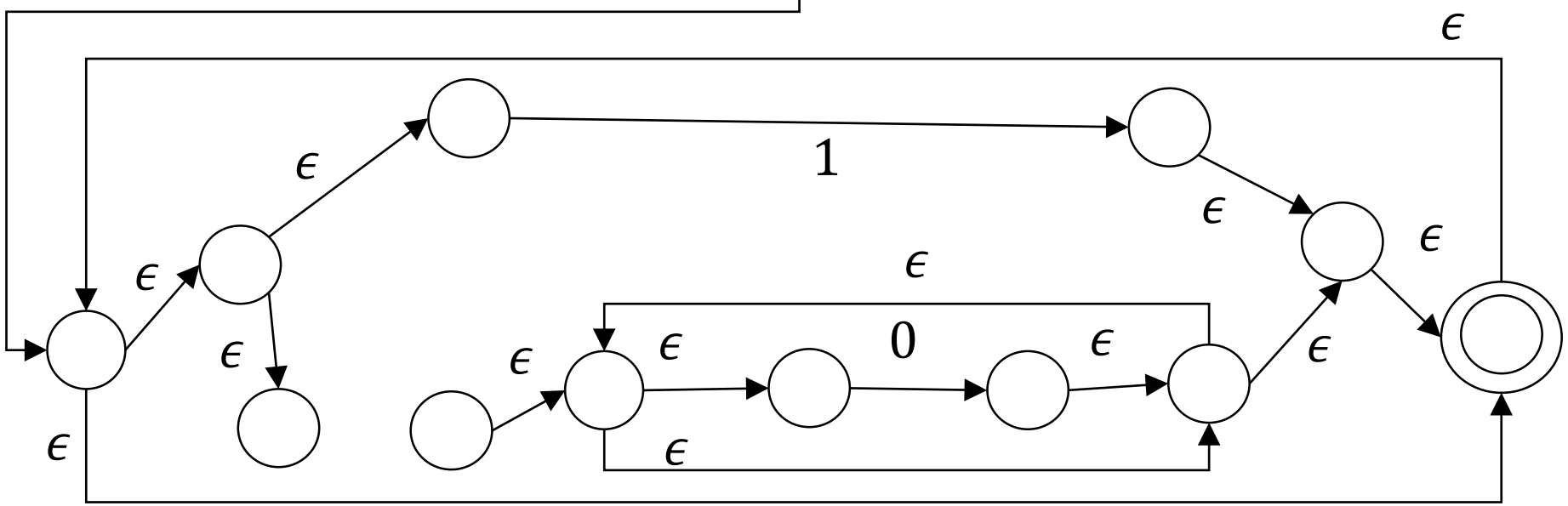


4. 有限オートマトンと正規表現の等価性 (19)

例：正規表現$0^*(1 + \emptyset 0^*)^*$が表現する言語を認識するϵnfa

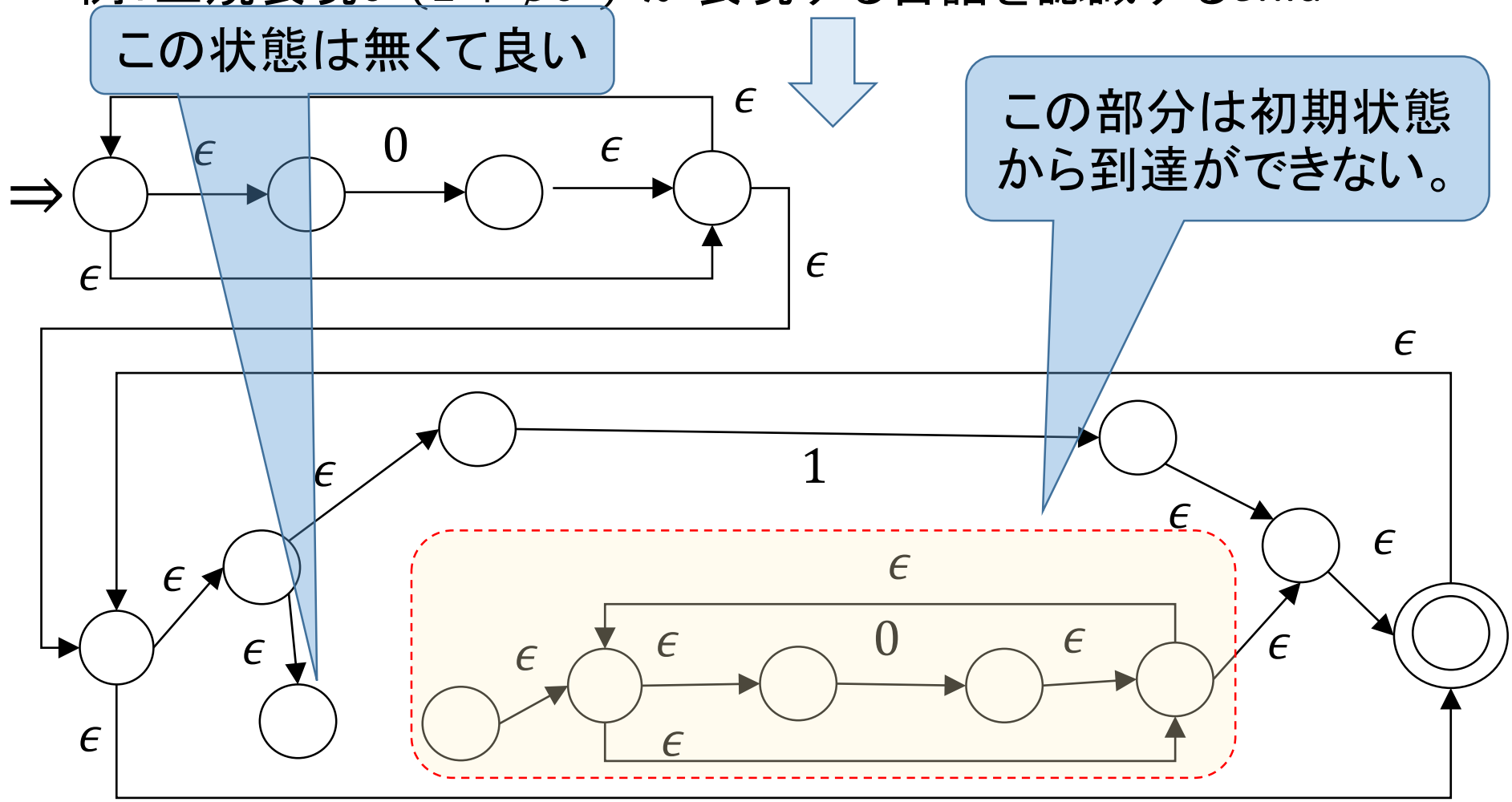


完成…



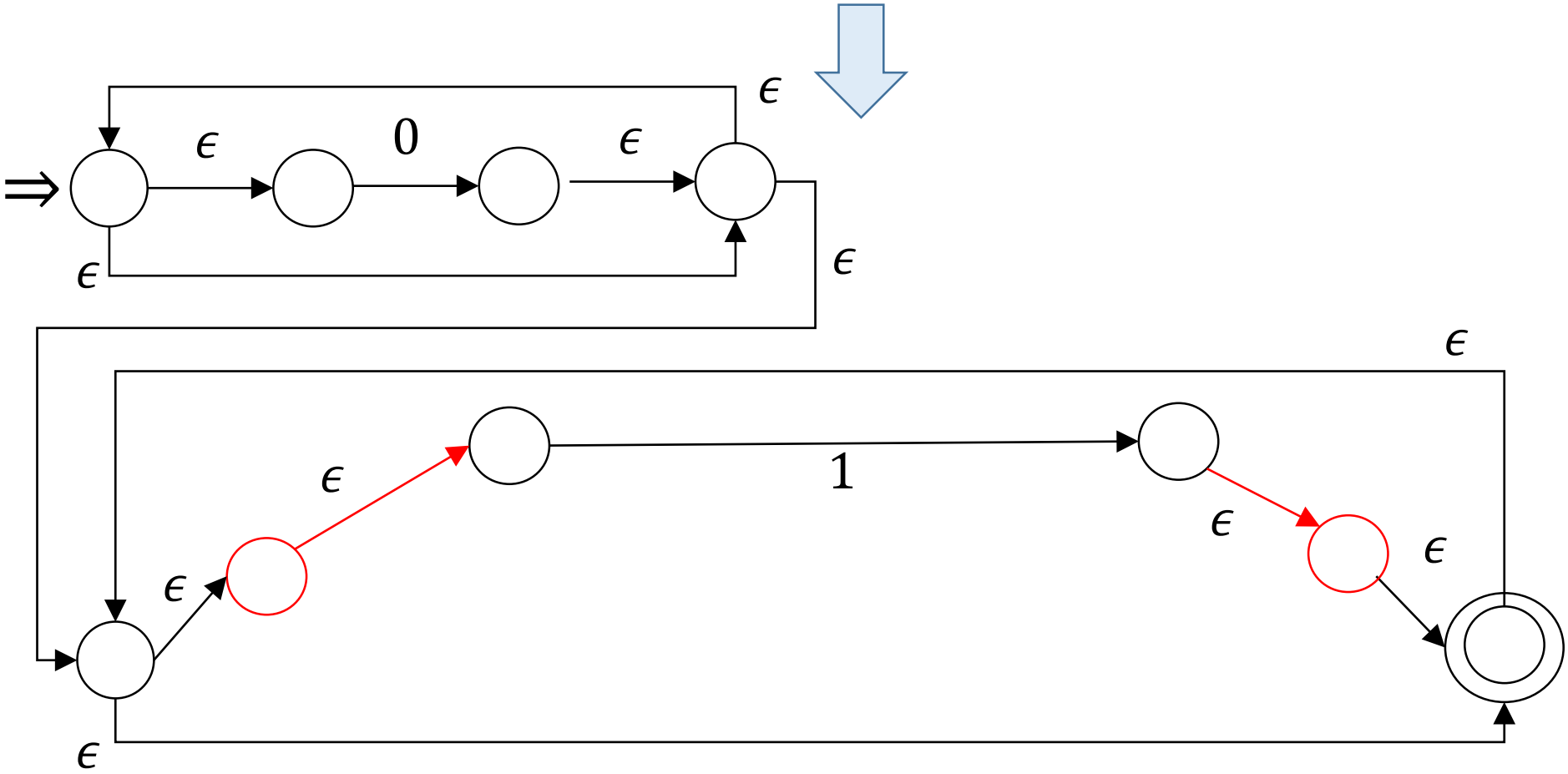
4. 有限オートマトンと正規表現の等価性 (19)

例：正規表現$0^*(1+\emptyset 0^*)^*$が表現する言語を認識するϵnfa



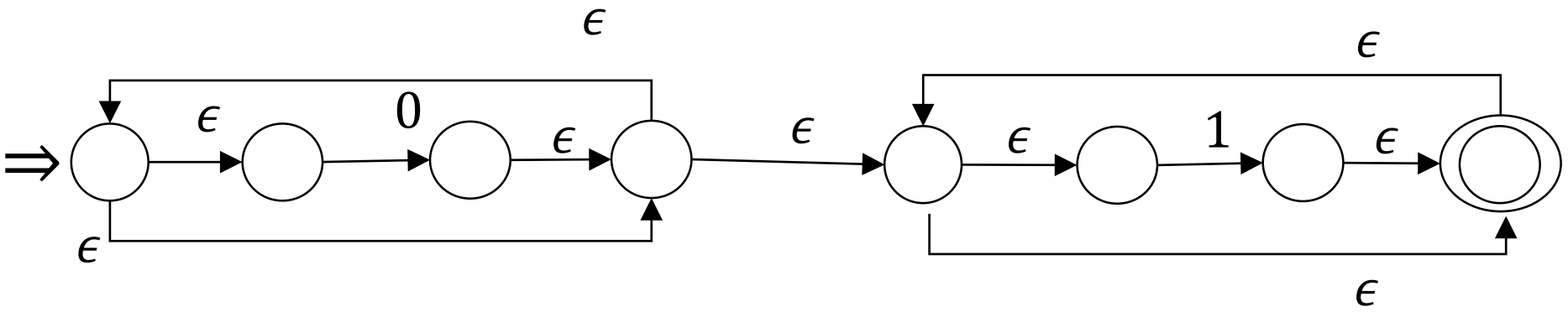
4. 有限オートマトンと正規表現の等価性 (20)

例：正規表現$0^*(1 + \emptyset 0^*)^*$が表現する言語を認識するϵnfa



4. 有限オートマトンと正規表現の等価性 (21)

例：正規表現$0^*(1+\emptyset 0^*)^*$が表現する言語を認識するϵnfa



そもそもよく考えると、$L(0^*(1+\emptyset 0^*)^*) = L(0^*1^*)$が成立している。

問題

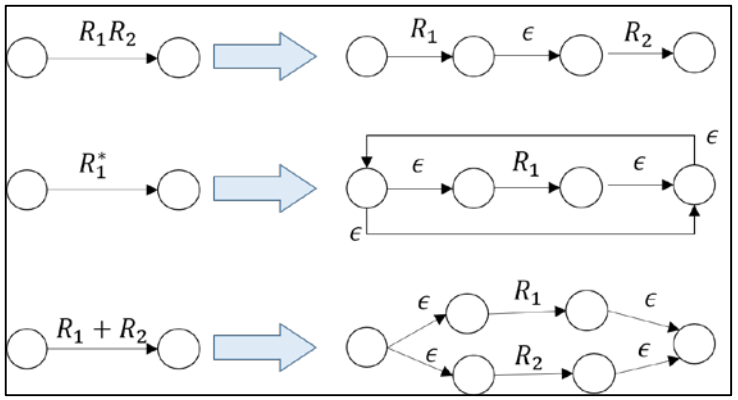
以下の正規表現で表される言語を認識するϵnfaの状態遷移図を書け。ただし、アルファベットは{0,1}とする。

① $(0+1)^*000(0+1)^*$

② $\left((00)^*(11)\right)+01\Big)^*$

③ $\emptyset^*$

Hint :



問題

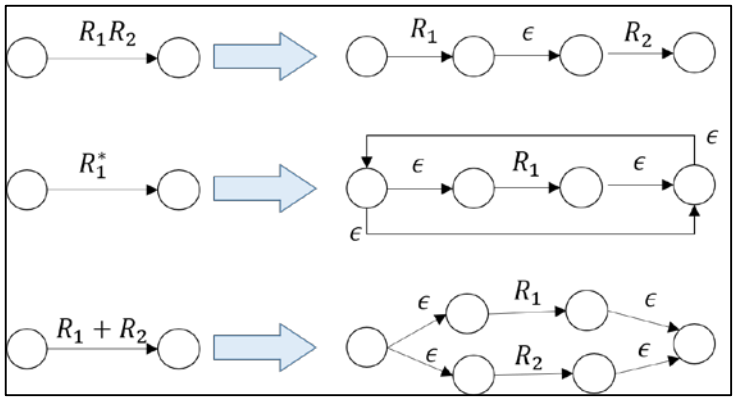
以下の正規表現で表される言語を認識するϵnfaの状態遷移図を書け。ただし、アルファベットは$\{a, b\}$とする。

① $(a(abb)^*) + b$

② $\big(a(a^*)\big) + (ab(ab)^*)$

③ $(a + b(b)^*)a(a)^*b(b)^*$

Hint：



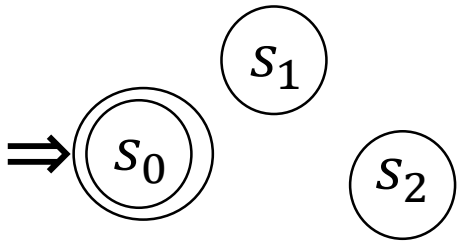
4. 有限オートマトンと正規表現の等価性 (22)

補題2.4 ($L(\text{nfa}) \subseteq L(\text{正規言語})$)　言語$L$がnfa Nで認識されるなら、Lはある正規表現Rよって表わせる。

（証明の概要）

与えられたnfa $N = (K, \Sigma, \delta, s_0, F)$の正規表現を、より矢の数が少ないnfaの正規表現に帰着させていく。

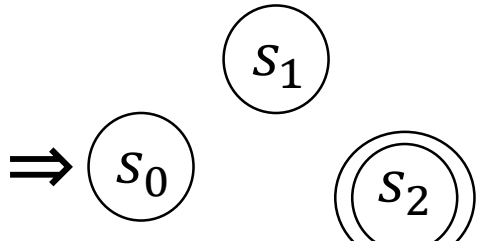
nfa Nに対応する正規表現を$Reg(N)$で表す。



まず、矢が一つもないnfaは、

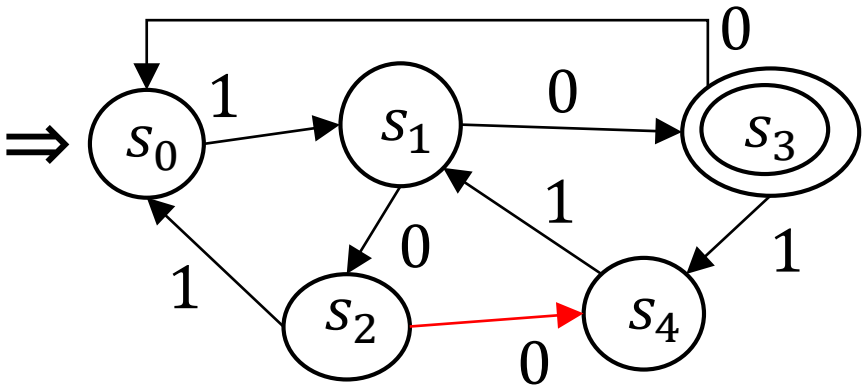
・$s_0 \in F$のとき、$Reg(N) = \epsilon$

・そうでないとき、$Reg(N) = \emptyset$　である。

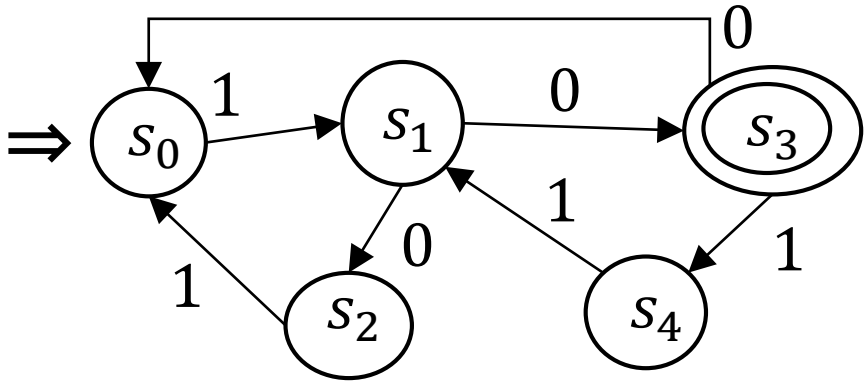


4. 有限オートマトンと正規表現の等価性 (23)

与えられたnfa $N = (K, \Sigma, \delta, s_0, F)$の矢の一つに注目する。



この矢を一つ取り除いたnfa をMとする。

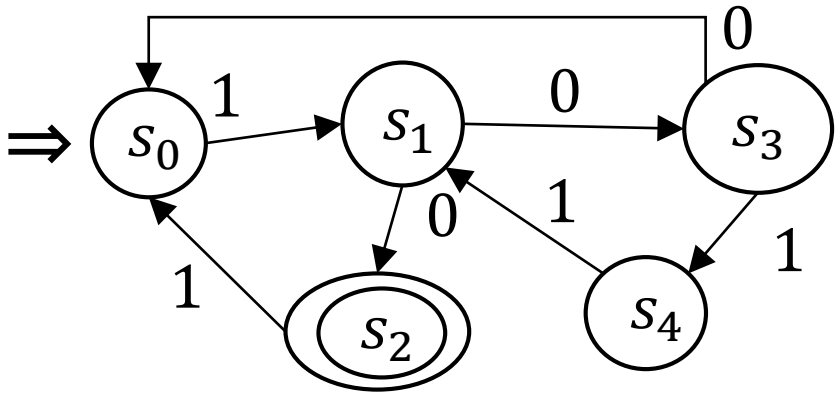


４. 有限オートマトンと正規表現の等価性 (２４)

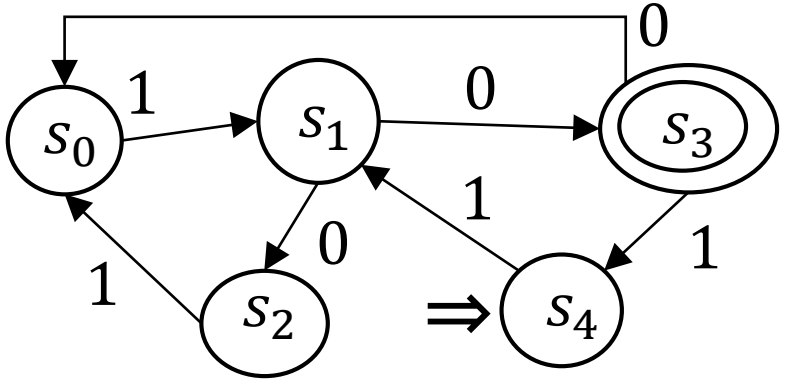
Mの初期状態をvに受理状態集合をVに変更したnfaを、$M(v,V)$とする。 つまり$M = M(s_0, F)$

例えば:

$M(s_0, \{s_2\})$

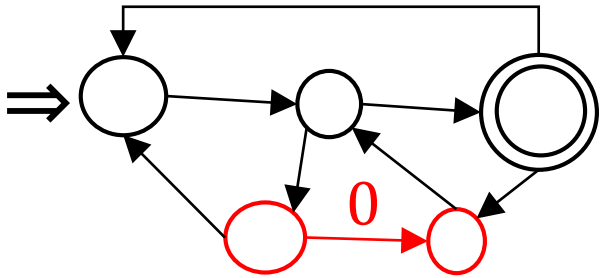


$M(s_4, F)$
$= M(s_4, \{s_3\})$

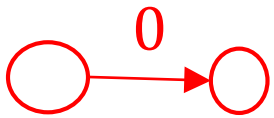


4. 有限オートマトンと正規表現の等価性 (25)

N:

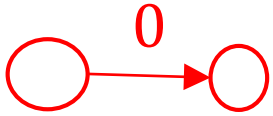


で認識される言語は、



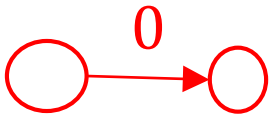
を1回も通らないで認識される言語

に



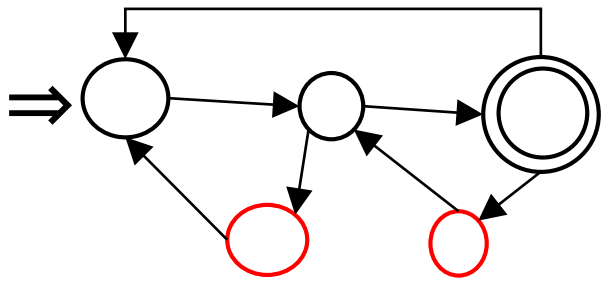
を1回以上通って認識される言語を加えたものである。

4. 有限オートマトンと正規表現の等価性 (26)

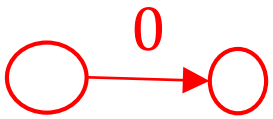


を1回も通らないで認識される言語

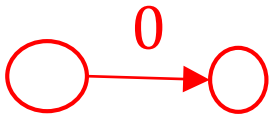
とは、Nから  を除いた、

M: ⇒  で認識される言語である。

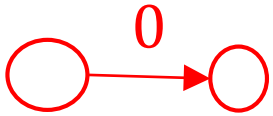
4. 有限オートマトンと正規表現の等価性 (27)



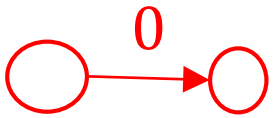
を1回以上通って認識される言語は、



を1回通って認識される言語と、



を2回通って認識される言語と、

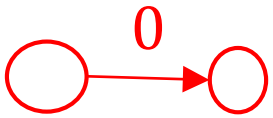


を3回通って認識される言語と、・・・・・

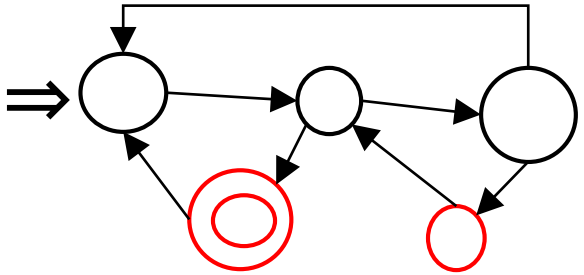
・・・・・・

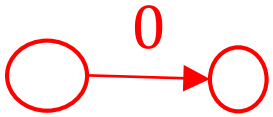
を全て合わせたものである。

4. 有限オートマトンと
正規表現の等価性 (28)

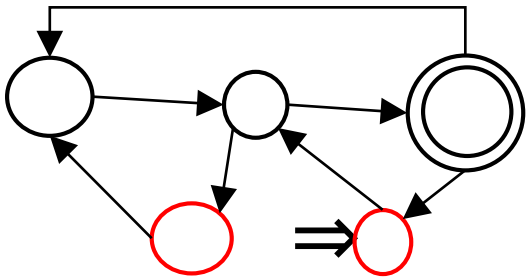


を1回通って認識される言語は、

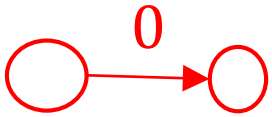

で認識される言語と、



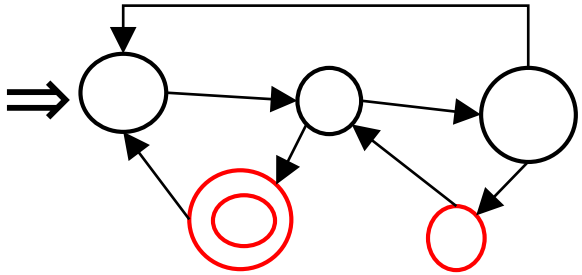
で認識される言語と

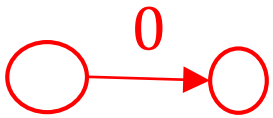

で認識される言語を
連接したものである。

4. 有限オートマトンと正規表現の等価性 (29)

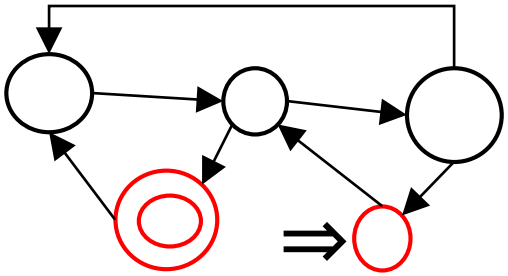


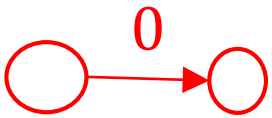
を2回通って認識される言語は、


で認識される言語と、

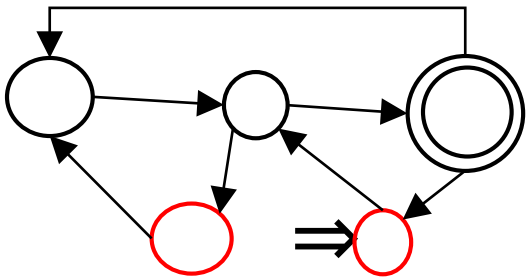


の言語と

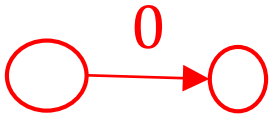

の言語と



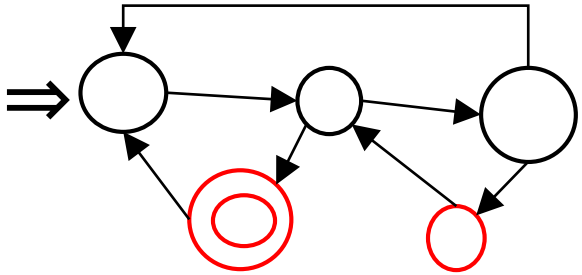
の言語と

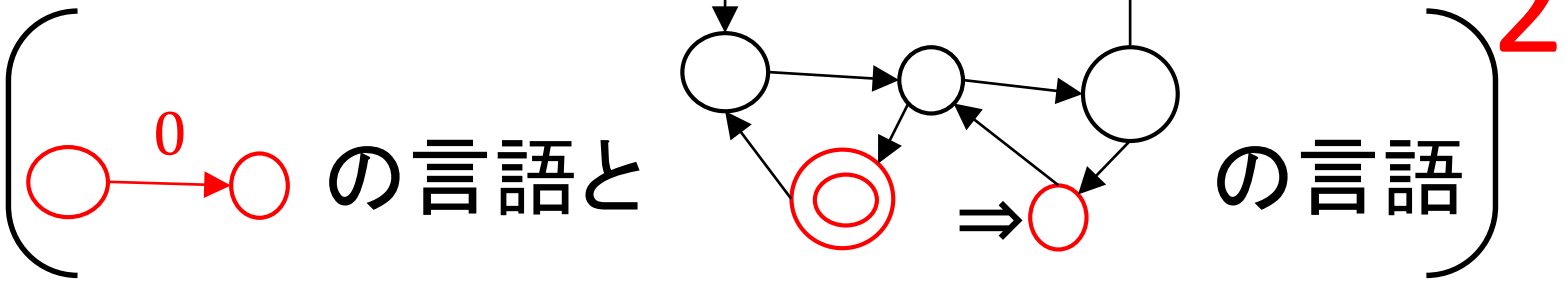

の言語を連接したもの。

4. 有限オートマトンと正規表現の等価性 (30)

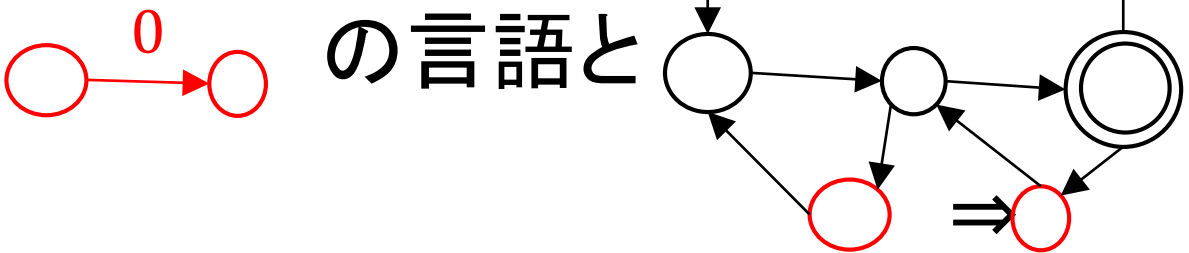


を3回通って認識される言語は、


で認識される言語と、

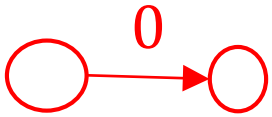


の言語と
の言語
と

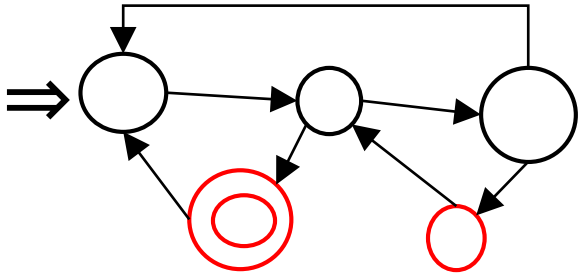


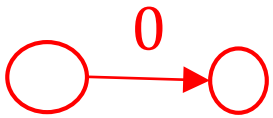
の言語と
の言語を
連接したもの。

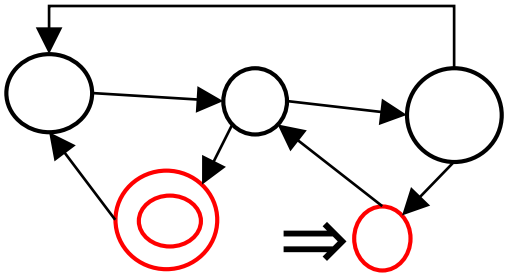
4. 有限オートマトンと 正規表現の等価性 (31)

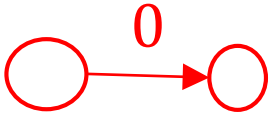


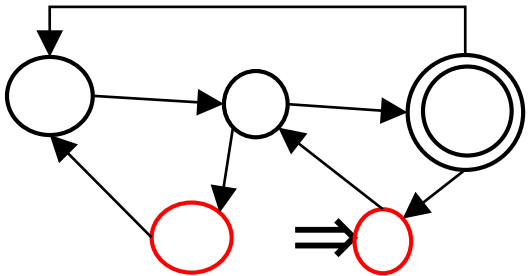
をn回通って認識される言語は、


で認識される言語と、

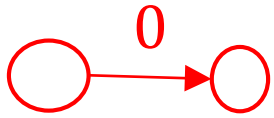
(

の言語と 
の言語)$^{n-1}$ と

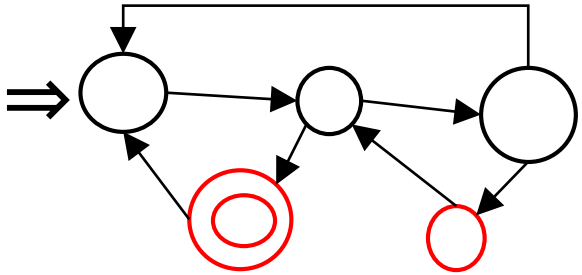


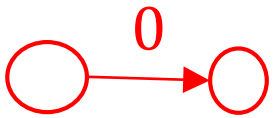
の言語と 
の言語を
連接したもの。

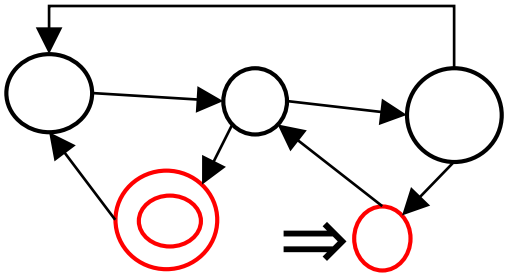
4. 有限オートマトンと正規表現の等価性 (32)

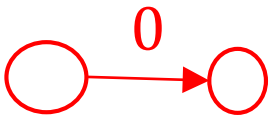


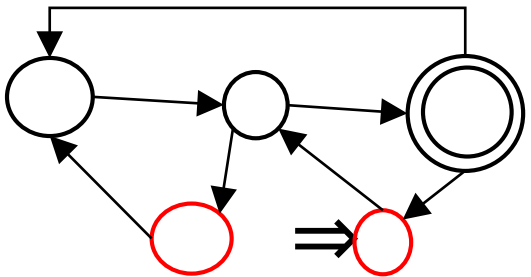
を1回以上通って認識される言語は、


で認識される言語と、

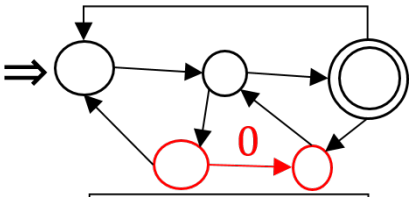
(

の言語と 
の言語)* と

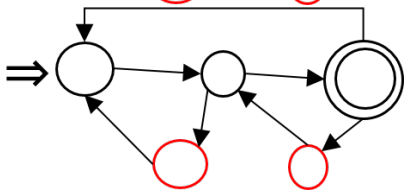


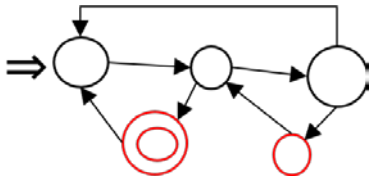
の言語と 
の言語を連接したもの。

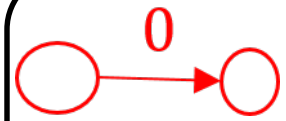
4. 有限オートマトンと正規表現の等価性 (33)

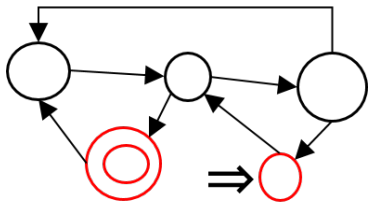
結局、 

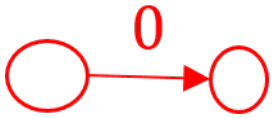
で認識される言語は、

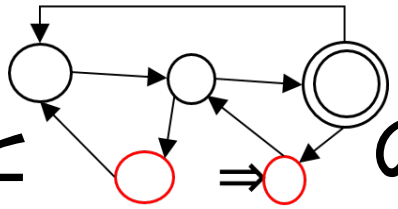

で認識される言語と、

{ 
の言語と

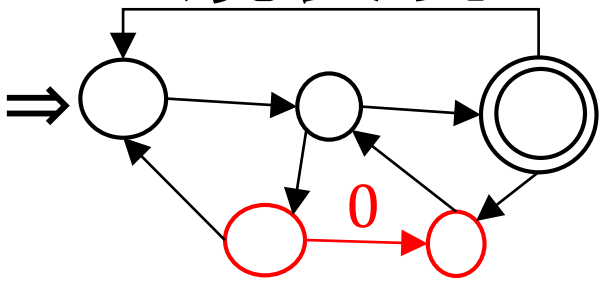
(

の言語と 
の言語)* と

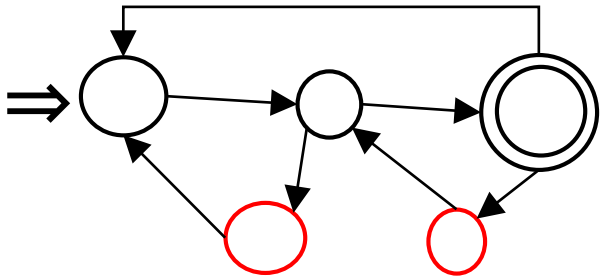


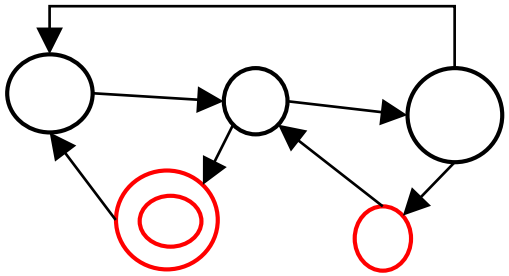
の言語と 
の言語の連接 } の足し算

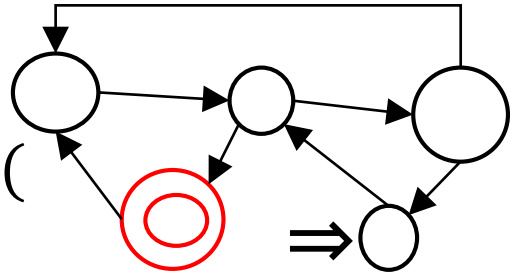
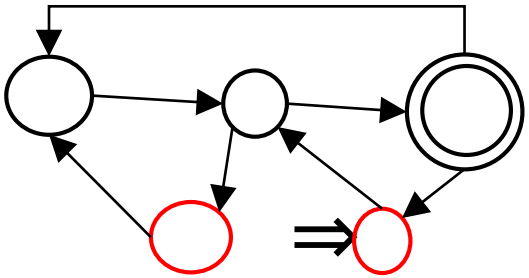
4. 有限オートマトンと正規表現の等価性 (34)

よって、 $Reg($ 

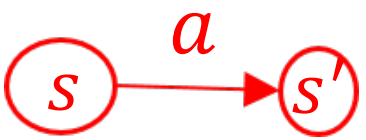
$)$

$= Reg($  $)$

$+Reg($  $)$

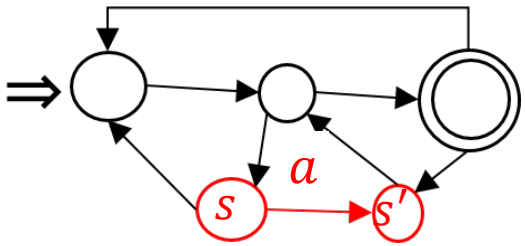
$\cdot [0 \cdot Reg($  $)]^* \cdot 0 \cdot Reg($  $)$

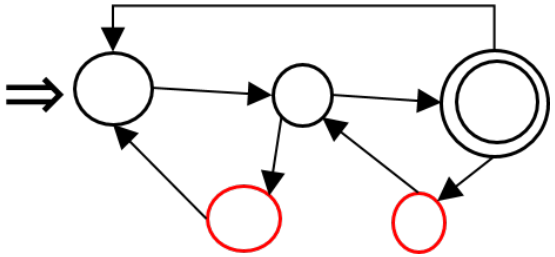
4. 有限オートマトンと正規表現の等価性 (35)

書き直すと、　　（状態の名前を変えて、 

とする）

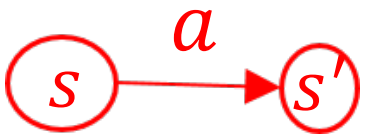
$$Reg(N)$$
$$= Reg\big(M(s_0, F)\big)$$
$$+Reg\big(M(s_0, s)\big) \cdot \big[a \cdot Reg\big(M(s', s)\big)\big]^* \cdot a \cdot Reg\big(M(s', F)\big)$$

ここで、Nは ⇒ 

Mは ⇒ 

Mの初期状態をvに受理状態集合をVに変更したnfaを、
$M(v, V)$とする。 つまり$M = M(s_0, F)$
また、“0”を”a”と書いた。

4. 有限オートマトンと正規表現の等価性 (36)

書き直すと、　　（状態の名前を変えて、 

とする）

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

Mは、Nよりも矢の数が1本少ないので、

Nの正規表現を、矢の数が1つ少ないnfaの正規表現に帰着できた。

実は、上記の式はどんなnfaのどんな矢に対しても成立する。

よって、この式を何度も使うことで、最終的に矢の一つもないnfaの正規表現に帰着できる。

矢が一つもないnfaは、$s_0 \in F$のとき$Reg(N) = \epsilon$、そうでないとき$Reg(N) = \emptyset$　である。

よって、この方法で任意のnfaの正規表現が求まる。

4. 有限オートマトンと正規表現の等価性 (37)

よって、$L(\text{nfa}) \subseteq L(\text{正規言語})$が証明できた。

すでに、$L(\text{正規言語}) \subseteq L(\text{dfa}) = L(\text{nfa}) = L(\epsilon n\text{fa})$は証明しておいたので、結局、$L(\text{dfa}) = L(\text{nfa}) = L(\epsilon n\text{fa}) = L(\text{正規言語})$

が証明できた。

定理：faと正規表現は等価である。即ち、faで認識できる言語は正規表現で生成できるし、逆もまた真である。

正規言語：有限オートマトンで認識できる（＝正規表現を持つ）言語

5. 有限オートマトンの能力の限界 (1)

有限オートマトン・正規表現は、様々な場面・用途で使われており重要である。

しかし、言語表現能力は低い。

例えば、カッコ文 “((( )))”が 左カッコ”(”の数　＝　右カッコ”)”の数という性質を持つかどうかを確かめたいとする。

”(“を0, “)”を1として、言語$\{0^n 1^n \mid n \geq 0\}$ はfaで認識できるか？

定理:　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。

証明は次のページ　⇩

5. 有限オートマトンの能力の限界 (2)

(証明)　鳩ノ巣原理を使う。

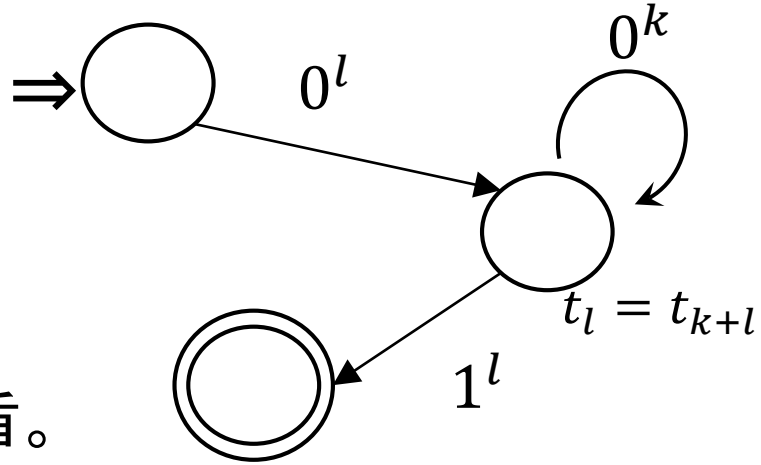
言語$\{0^m 1^m \mid m \geq 0\}$がfa　Mによって認識できたとする。
Mの状態数をnとする。

状態t_iを、$t_i := \delta(s_0, 0^i)$ で定義する。

状態数がnなので、$t_0, t_1, \cdots, t_{n+1}$の中に少なくとも2個の同じ状態が含まれる。これらをt_lとt_{l+k} とする。（$t_l = t_{l+k}$）

Mは列$0^l 1^l$を受理する。
よって、$\delta(t_l, 1^l)$は受理状態。

しかし、$t_l = t_{l+k}$なので、
$\delta(t_l, 1^l) = \delta(t_{l+k}, 1^l)$。
よって、$0^{l+k}1^l$も受理される。これは矛盾。



5. 有限オートマトンの能力の限界 (3)

例2.13 : 言語$\{xx \mid x \in \{0,1\}^*\}$はいかなるfaによっても認識できない。

(証明)

同様に鳩ノ巣原理で証明できる。
この言語を認識できるfa Mを考える。Mの状態の数をnとする。
M は10^n10^nを認識するので、状態遷移関数δを用いて、
$t_0 = s_0, t_1 = \delta(s_0, 1)$、$t_i = \delta\left(s_0, 10^{i-1}\right)$とする。
$t_0, t_1, \cdots, t_n$のうち、同じ状態が少なくとも2個ある。
それをt_lとt_{l+k} とする($k \geq 1$)。

i) $l = 0$のとき: $\delta(t_0, 10^n10^n) = \delta(t_k, 10^n10^n)$は受理状態。
これでは、$10^{k-1}10^n10^n$を受理するので矛盾。

ii) $l \geq 1$のとき: $\delta\left(t_l, 0^{n-l+1}10^n\right) = \delta\left(t_{l+k}, 0^{n-l+1}10^n\right)$が受理状態。この場合、$10^{l+k-1}0^{n-l+1}10^n = 10^{n+k}10^n$が受理され矛盾。
よって、この言語はfaでは認識できない。